



The information contained herein is for the use of employees of Bell Laboratories and is not for publication (see GEI 13.9-3)

Title: File System Structures for Real-Time Applications

Date: April 20, 1978

Other Keywords: Time Sharing  
UNIX  
MERT  
Asynchronous I/O

TM: 78-3114-5

Author(s)  
H. Lycklama

Location  
HO 1G-317

Extension  
3212

Charging Case: 39394  
Filing Case: 39394-11

ABSTRACT

File system structures have been designed for various versions of the UNIX\* and MERT operating systems over the past few years. Each structure was designed to be efficient in a particular environment, i.e. the nature of the application dictated the design.

The structure of the MERT operating system enables one to write a new file manager process with minimum impact on other parts of the system. Only the file system utility commands need to be rewritten to deal with different file system structures. The original file manager written for the MERT system made use of *extents* to allocate contiguous space to files. This makes these files optimal for real-time applications. For time-sharing applications, a file structure like the original 16-bit UNIX file structure is more appropriate.

Recently a new version of the file system structure was done for Version 7 UNIX using a 32-bit byte offset pointer to achieve very large files and file systems. Subsequent to this a new 32-bit file system was designed and built for the MERT system. The new file system structure has characteristics which make it efficient for both time-sharing and real-time applications. This paper describes the salient features and shortcomings of each of the four file system structures.

\* UNIX is a Trademark of Bell Laboratories

Pages Text: 8	Other: 15	Total: 23
No. Figures: 0	No. Tables: 0	No. Refs.: 6



**Bell Laboratories**

subject: **File System Structures for Real-Time Applications**

Case: **39394**

File: **39394-11**

date: **April 20, 1978**

from: **H. Lycklama**

**HO 3114**

**1G-317 x3212**

TM: **78-3114-5**

### *ABSTRACT*

File system structures have been designed for various versions of the UNIX<sup>\*</sup> and MERT operating systems over the past few years. Each structure was designed to be efficient in a particular environment, i.e. the nature of the application dictated the design.

The structure of the MERT operating system enables one to write a new file manager process with minimum impact on other parts of the system. Only the file system utility commands need to be rewritten to deal with different file system structures. The original file manager written for the MERT system made use of *extents* to allocate contiguous space to files. This makes these files optimal for real-time applications. For time-sharing applications, a file structure like the original 16-bit UNIX file structure is more appropriate.

Recently a new version of the file system structure was done for Version 7 UNIX using a 32-bit byte offset pointer to achieve very large files and file systems. Subsequent to this a new 32-bit file system was designed and built for the MERT system. The new file system structure has characteristics which make it efficient for both time-sharing and real-time applications. This paper describes the salient features and shortcomings of each of the four file system structures.

### *MEMORANDUM FOR FILE*

#### **1. Introduction**

A file system provides the user with an interface to data on secondary storage as well as the structures necessary to map an offset into a file to a disk address. The structure of a file system has enormous impact on the efficiency of I/O transfers to/from a file and therefore on the applications which run on the system. The crucial aspects of a file structure are determined by the file map and the mode of maintaining free block lists.

There are basically three ways of defining a file map:

- linked list of blocks
- indexed list of blocks
- extents of blocks

\* UNIX is a Trademark of Bell Laboratories.

A linked list of blocks uses one word in each block to point to the next block in the file. This file structure is suitable for sequential access to a file in the forward direction but it is not suitable for random I/O. The indexed structure makes use of a list of block pointers to point to the blocks which make up the file. If the list of blocks is greater than a fixed number, an indirect block of block pointers is used. This file system structure permits both sequential and random I/O transfers. However random I/O may require at least two disk accesses to access an arbitrary block in the file. The use of extents to define the blocks in a file enables the allocation of contiguous files with the added advantage of random I/O with only one disk access. An extent is a two-element structure in which the first element indicates the starting block number and the second element indicates the number of consecutive blocks. These features are often requirements for real-time systems.

The means of maintaining free block lists is another crucial aspect of a file system design. There are four basic strategies for doing this:

- maintain bit map in memory
- maintain bit map on disk
- use linked list of free blocks
- use list of free extents.

A bit map requires one bit of storage per allocatable block. A bit set to one indicates a free block. The use of a bit map in core is useful and only practical for small file systems. This allows the allocation of a block or the freeing of a block without a disk access. For larger file systems, it is necessary to maintain the bit map itself on disk. Parts of the bit map may be cached in memory. For the random deletion of single blocks from a file, a disk access is required to update the bit map. Thus this is not a good strategy for maintaining free block lists for time-sharing systems. The use of a linked list of free blocks can be used for a time-sharing system by keeping a cache of free blocks in core. When the in-core list is exhausted, a disk access is required to get a new cache of free blocks. When the in-core list is filled, one disk access is required to empty the list. For real-time systems, it is more efficient to maintain a list of free extents in memory. This requires no extra disk accesses for the allocation or de-allocation of blocks and also ensures the ability to allocate a number of consecutive blocks.

This memorandum describes various file structures which have been designed and built for the MERT real-time system. For the sake of completeness, the salient features and shortcomings of each of the four file system structures which have been implemented on the UNIX and MERT systems are described. The contents of the superblock, of the on-disk inode and of the in-core inode are described for each file system. The map of a file (inode) and the technique for maintaining free blocks are discussed in detail. Two of the file systems have been implemented on the UNIX system, whereas three of the file systems have been implemented on the MERT system. The C structures used for the superblock, the on-disk inode and the in-core inode are listed in Appendices A-L. For the MERT file systems, inodes on disk are often referred to as vtoc entries (volume table of contents).

This memorandum gives only an overview of the characteristics of each one of these file systems and does not give detailed usage or timing statistics. The reader is assumed to be familiar with the UNIX time-sharing system [1] and the basic structure of the original UNIX 16-bit file system [2]. The MERT system is described in a previous memorandum [3].

## 2. Directory Structure

The overall directory structure of the file systems is described first as this is identical for all four file systems. The file system naming convention is completely hierarchical. A file system has a root inode which describes a directory. In fact a directory is a special type of file which contains the names and inode pointers of other files and directories. A directory entry is a 16-byte entry consisting of a two byte inode number and up to fourteen characters specifying the name of the file. The root inode is referred to by "/". A directory "usr" in the root directory is referred to as "/usr". A directory "bin" in "/usr" is referred to as "/usr/bin" and so on. Directories may go to an arbitrary level. Further details of the directory structure may be found in a previous memorandum [2].

### 3. File System Layout

A file system consists of a number of contiguous 512-byte blocks. The first block of a file system (block 0) is reserved for a boot program. Block 1 is called the "superblock". The next N blocks are reserved for the "inodes" or the control blocks for all files in the file system. The number of inodes in a block depends on the size of one inode. It is different for each file system discussed. If an inode is 32 bytes in size, there are 16 inodes per block and thus 16N inodes in total. Thus a total of 16N files may be maintained on this particular file system. For UNIX file systems, the remainder of the blocks in a file system are reserved for data blocks in files, directories and free lists. For MERT file systems, a number of blocks beyond the inode blocks are used for maintaining bit maps which describe an up-to-date free block list.

The superblock contains a roadmap of how to access the rest of the file system. For each type of file system, it contains at least the size of the ilist (list of inodes) in blocks and the size of the total file system in blocks. It also contains a list of free inodes available for the creation of a new file. For UNIX file systems, the superblock contains a list of free blocks. For the MERT 16-bit file system, the superblock contains a list of free extents. The superblock also contains a number of other miscellaneous items. These are different for each file system type.

The various file systems are now described in further detail in the order in which they were historically developed. They were all initially developed for PDP-11 computers for the various disk storage devices available at the time.

### 4. UNIX 16-Bit File System

The file structure developed for the original UNIX system written in the C language for the PDP-11/45 computer was designed to handle file systems for 16-bit mini-computers and disk storage devices with a limited number of disk blocks per device. A block number is an unsigned 16-bit value. Thus up to  $2^{16}$ , i.e. 65536 blocks may be addressed in one file system. The free blocks in the file system are put on a free chain of blocks with each block holding a list of 100 free blocks. The order of blocks on the free list may be optimized to minimize disk latency when reading blocks from a file. The free list is self-consuming as it is used up. All of the items in the superblock structure are appropriately commented in Appendix A. Up to a hundred free blocks are maintained in the in-core superblock. The actual number is given by  $s\_nfree$ . The items at the end of the superblock were added subsequent to the design of the original UNIX file system. The total number of free blocks on the file system  $s\_tfree$  and the total number of free inodes on the file system  $s\_tinode$  are maintained and are used by special user-level software to determine whether a complete file can be written before an attempt is made to write a file and to keep track of the total disk blocks free at any one time.

The inode structure on disk is given in Appendix B. It is 32 bytes in length. Here the inode is referred to as a *v*toc, volume table of content entry, this being the MERT system naming convention. Each item in the inode is named with a leading 'v' rather than with a leading 'i' to avoid confusion with the in-core inode structure names. Note that a character is devoted to each of the items, user id, group id and the file link count. The size of the file is described by 24 bits. Block numbers are specified by  $v\_addr[i]$  where a small file is less than eight blocks long and a large file uses these as pointers to blocks containing block numbers. This gives a maximum file size of  $8*256*512$ , i.e. 1M bytes. A variation of this has been implemented to achieve "huge" files. This uses  $v\_addr[7]$  to point to an indirect block which in turn points to 256 double indirect blocks. This allows the addressing of more than 32M bytes. However, the 24-bit size field in the inode limits the largest file to 16M bytes. Thus huge files may require up to three disk accesses to retrieve a random block number in a file.

The in-core inode structure is included for completeness in Appendix C. The structure is almost identical to the on-disk structure with additional bookkeeping parameters. The  $i\_count$  item is maintained to keep track of the number of opens. The final close causes the inode to be written back to disk.

### 5. MERT 16-Bit File System

The original MERT file system was designed to be efficient for real-time applications. However, the file system had to be suitable for time-sharing applications as well. Time-sharing applications require

that files be both dynamically allocatable and growable. Real-time applications often require that files be large and possibly contiguous; dynamic allocation and growth are usually not required. Contiguous files are efficiently described by a file map entry which consists of starting block number and number of consecutive blocks (a two-word extent). All files are allocated by extents. Files can grow or shrink in size by adding/growing extents or deleting/truncating extents, respectively. A file can have up to 27 extents.

The list of free blocks is kept in the superblock as a list (see Appendix D) of the 64 largest extents of contiguous free blocks. Blocks for files are allocated and freed from this list using an algorithm which minimizes file system fragmentation. When allocating blocks to a file, a fixed number of blocks (set at system generation time) is found in the free list from an entry which best matches the required number of blocks. When freeing blocks, the blocks are merged into an existing entry in the free list if possible, otherwise placed in an unused entry in the free list or, failing this, they replace an entry in the free list which contains a smaller number of free blocks.

Entries which are being freed or allocated are also added to an update list in the in-core superblock in *s\_update[]*. When this update list becomes full (*N\_UPDATE* entries), further modifications to the superblock are temporarily blocked while these update entries are used to update a bit map which resides on secondary storage at the end of the ilist. The number of blocks reserved for this bit map is sufficient to contain one bit for each block in the file system, *s\_size*. If the in-core free list in the superblock should become exhausted, the bit map is consulted to recreate the 64 largest entries of contiguous free blocks. The nature of the file system and the techniques used to reduce file system fragmentation ensure that this is a very rare occurrence.

The inode (or vtoc) structure as stored on disk is given in Appendix E. Each inode is 128 bytes long. This allows only 4 inodes per block. There are a number of differences between the UNIX 16-bit file system inode and the MERT 16-bit file system inode. An invocation count, *v\_invoc*, is maintained in the inode to determine the uniqueness of the file. This value is incremented when the inode is de-allocated. Two complete words, i.e. 32 bits, are used to specify the size of a file. The maximum file size is actually  $512 * 2^{16}$ , i.e. 32M bytes. A file may actually consist of up to 27 extents of contiguous blocks. A different file type is described by the ICONT mode bit. This is used in place of the ILARG bit in the UNIX 16-bit file system inode. If the ICONT bit is set, only one extent may be used to describe the file map. This guarantees that once the file is open, any part of the file may be read in one disk access. The contiguous file must be created with a given number of consecutive blocks. These blocks are not given back to the free list until the file is deleted even though only a fraction of the blocks are written into.

The in-core inode structure for the MERT 16-bit file system is shown in Appendix F. Because of size restrictions, the complete on-disk inode is not stored in core. Only one extent is kept in the in-core inode, providing a window into the file. The relative block number of the first block in this extent is given by *i\_stblk* and the extent number in core is given by *i\_extno*. A block within this extent can be accessed in one disk transfer. Blocks not included in this extent require two disk I/O's to access. The last extent used remains in core, thus acting as a cache entry. For real-time applications where more than one disk access to access a random block or number of blocks cannot be tolerated, the file may be moved into a contiguous area of disk and thus be defined by one extent.

When a non-contiguous file is closed, unused blocks at the end of the file, beyond the size of the file, are returned to the free list. Contiguous files use only one extent. The blocks described by this extent are only freed when the last link to the file is removed even though the size of the file as specified by *i\_size* is less than indicated by the number of blocks allocated.

Very active file systems consisting of many small time-sharing files may be compacted periodically by a utility program to minimize file system fragmentation still further. File system storage fragmentation actually only becomes a problem when a file is unable to grow dynamically, having used up all 27 extents in its file map entry. Normal time-sharing files do not approach this condition. Fragmentation becomes a problem when the in-core free list of extents becomes sparse, with each free extent only representing a few contiguous free blocks. When the in-core list of extents is exhausted, it is replenished from the up-to-date bit map on disk.

## 6. UNIX 32-Bit File System

With the advent of larger disks and requirements for larger data bases, it became cumbersome to deal with the multiple file systems which are required on one disk since a 16-bit block number can only address  $2^{16}$  blocks. A 160M byte disk with 320K blocks would require five file systems to address the complete disk. To treat this disk as one file system requires more bits in the block number. A recent effort at porting the UNIX system to a 32-bit machine also required a new approach to the UNIX 16-bit file system structure. Thus a new 32-bit file system was designed for Version 7 UNIX [4]. This file system is described briefly here since the new MERT 32-bit file system is based on it.

The new format UNIX file system is referred to as a 32-bit file system since a two-word (16 bits each) data type is used to provide a byte offset into a file. Actually block numbers are only 24 bits long. Thus a file system may contain up to  $2^{24}$ , i.e. 16M blocks or 8B bytes of data. However the 32-bit size word in the inode, discussed later, limits file size to 4B bytes. The superblock for this file system shown in Appendix G, bears a strong resemblance to the superblock for the 16-bit UNIX file system. The main difference is that 32 bits are used to store block numbers rather than 16 bits. The total number of free blocks in the file system is also stored as a 32-bit data type. The total number of free blocks kept in core is only 50. The free blocks are still maintained in a linked list, with each block holding 50 free blocks.

The inode structure maintained on disk is shown in Appendix H. Each inode is 64 bytes long, twice as long as the original UNIX 16-bit file system inode. This allows 8 inodes per block. The sizes of most of the items in the structure have changed. Note that the possible number of links in *di\_nlink* has increased to  $2^{16}-1$ . The group and user id's are now represented by a 16-bit word. The size of a file is now represented by 32 bits. There is no longer any distinction made between small and large file types. The array *di\_addr[40]* contains 13 24-bit block numbers. The first 10 of these are the first 10 block numbers in the file. The eleventh entry is a pointer to a block of 128 indirect pointers, the twelfth a pointer to a block of double-indirect pointers and the thirteenth a pointer to a block of triple-indirect pointers. Thus a file may contain somewhat more than  $2^{21}$  blocks, i.e. 2M blocks or 1 billion bytes. To access a random block in an enormous file (with triple-indirect pointers) may require up to four disk accesses.

The in-core inode structure is shown in Appendix I. The difference between the structure stored in memory compared to the on-disk structure is that the block numbers have been unpacked from 24 bits to 32-bit values to make it easier to obtain a block number in the structure. The last logical block read is also maintained in the in-core inode structure for the implementation of read-ahead. This makes the size of the in-core inode much larger than that required for the UNIX 16-bit file system. One inode occupies 74 bytes of memory. This is expensive for systems with a large number of open files and a limited system address space. Note that the file type is now encoded in 3 bits of the inode field to allow for more different file types. There is no longer an allocated bit. An unallocated inode has a value of 0 for this field.

The new 32-bit file system structure served the purpose of providing a general file system suitable for 16-bit as well as 32-bit mini-computers. Maintaining 10 blocks in the inode compared to 8 blocks in the 16-bit file system structure for small files is an improvement. In the new file format, each of the first 10 blocks can always be retrieved with one disk access. There is no modification required to the inode to grow a small file into a large file. This file system structure served as the base for building the new 32-bit file system for the MERT system.

## 7. MERT 32-Bit File System

The advent of new and larger disks and the prospect of portability of the MERT system to 32-bit machines required the redesign of the original MERT file system to a suitable structure just as for the UNIX system. Experience with the MERT 16-bit file system structure demonstrated that the need for a development environment as well as a running environment for real-time applications was an important consideration in designing a file system. There are applications where an off-line compaction of a file system is not feasible. The number of applications which require true contiguous files is not great.

\* Changes have been proposed to the file types by G. W. R. Luderer to make the new MERT file system types compatible with the new UNIX file system types. This requires 4 bits for the file type mask.

Thus a new file system was designed to make it more suitable for small time-sharing files and yet allow for the allocation of large contiguous files. Compaction of such a file system is not required.

The superblock for the MERT 32-bit file system as shown in Appendix J bears a strong resemblance to that of the UNIX 32-bit file system superblock. The structure items with similar names serve the same purpose as those in the corresponding UNIX file system superblock. The maintenance of the block free list is handled in a different fashion from that in the corresponding UNIX file system. A bit map for all blocks in the file system is maintained on disk beyond the ilist. A linked list of free blocks is generated from this bit map in a dynamic fashion. At the time of creation of the file system, NHWAT blocks are put on the linked free list with the interleave factor specified by *s\_m* in multiples of *s\_n*. The interleave factor is determined to minimize disk latency when accessing blocks in a new file in a sequential manner. The order on the free list determines the order in which blocks are allocated to a new file. For regular files, blocks are allocated from the linked list of free blocks. When this list is exhausted, new blocks are added to the chain dynamically from the bit map starting at *s\_nextblk* in an interleaved fashion. This value is updated when the blocks are allocated or de-allocated from the bit map. The total number of free blocks on the chain is maintained in *s\_cfree* at all times. As blocks are freed from files, they are added to the free chain but not marked free in the bit map.

Contiguous files are supported in a different manner in this file system structure. Blocks for contiguous files are allocated directly from the bit map. To avoid a linear search from the beginning of the bit map each time space is required, a pointer *s\_nextcon* is maintained, pointing to the last block number allocated to a contiguous file. Thus to allocate blocks to a contiguous file, the appropriate bit maps must first be read into memory. This is an expensive operation (in terms of disk I/O's) but does not have to be performed very frequently and certainly not in the critical path of a real-time task if use is made of pre-allocated contiguous files. The pointer *s\_nextcon* is updated each time blocks are allocated to or freed from a contiguous file. The block pointers, *s\_nextblk* and *s\_nextcon* are initialized by either the *mkfs* program or the *icheck* program.

The on-disk inode structure for a file in the MERT 32-bit file system is identical to that for the UNIX 32-bit file system (see Appendix K). It occupies 64 bytes, allowing 8 inodes per block. For regular files, the 13 addresses of 24 bits each in the *di\_addr[40]* array are used for the same purpose as the corresponding UNIX file system structure. For contiguous files and those files whose blocks are allocated by extents, only 12 24-bit numbers are stored. These correspond to 6 extents, where each extent is a starting block number and a number of contiguous blocks. The maximum file which can be described by an extent is  $2^{24}$  blocks or  $2^{33}$  bytes, which is larger than can be represented in the *di\_size* data field. Thus the maximum file size is  $2^{32}$  bytes, i.e. 4 billion bytes. This is larger than any secondary storage device currently available. The unused byte *di\_addr[39]* is used as the invocation count value in the MERT 32-bit file system in the same manner as the MERT 16-bit file system uses such an item.

The in-core inode structure given in Appendix L is similar to the in-core inode structure used by the corresponding UNIX file system. Two extra items are added to the structure. The *i\_invoc* field is used to determine the uniqueness of the file. It is incremented when the file is deleted. The *i\_use* field is used to determine the uniqueness of the in-core inode. When the *i\_count* value goes to 0, *i\_use* is incremented so that when this inode entry is reclaimed, the key to access the inode has been changed for security. Again the 13 addresses (24-bit) in the on-disk inode are unpacked to 13 32-bit addresses.

The file type field is 4 bits wide in the new MERT file system. Note that the mode bits indicate two "types" of contiguous files. When the file is of type IFEXT, blocks are allocated to the file by extents directly from the bit map. Up to six extents may be allocated. If the file is of type IFIXT, only one extent may be allocated to the file to ensure that it is a completely contiguous file.

## 8. Implementation of MERT File Systems

The file system is controlled completely by a file manager process. The file manager is a kernel mode process with I and D space separated to obtain enough virtual address space. Making the file manager a separate process eases the communication of other processes with the file manager. The file manager is the only process with knowledge of the detailed structure of the file system. The UNIX system supervisor in the MERT system has no need to deal with the file system structure. An RSX-11D supervisor process has been brought up to run on the MERT system to demonstrate the functional

separation of the independent processes. All supervisor processes communicate with the file manager process by means of messages. To prevent corruption of the file system, all incoming messages must be carefully validated. Each message includes a *capability* which is checked by the file manager process.

Internally the file manager is organized as a multi-tasking system, with each task handling one incoming message. This increases the total throughput of the file manager process. The task is created upon receiving a new message and destroyed when the final acknowledgement message is sent back to the sender. The tasks operate on a common data base and are not individually preemptible. Each task has a private data area as well as a common data area.

The file manager recognizes 25 different types of messages. Requests for reads or writes from/to a file must specify a starting block number and the number of bytes to transfer. If the request spans a number of file extents, the file manager process breaks the transfer up into a number of consecutive transfers. This feature is taken advantage of by the 'exec' system call in the MERT/UNIX supervisor. It is possible to transfer the complete executable file image into a segment with one I/O transfer, excluding the header block.

Two file system primitives have been added to the MERT file manager process to deal with contiguous files. These are reflected as MERT/UNIX system calls [5] as well as UNIX system user commands. The *falloc* primitive is provided to allocate space for a contiguous file. The size of the file to be created is given in bytes. The blocks allocated are not put back on the free list until the file is deleted. The second one provided is the *fmove* primitive. This is useful for combining all of the extents of a file (allocated by extents initially) into one extent. The file is not marked contiguous; thus it may be expanded dynamically at some later time. Further details on these and other file manager primitives are discussed in section C of the MERT Programmer's Manual [6].

## 9. Summary

Currently three different file system structures have been implemented for the MERT system. The details of the file system structure are controlled by the file manager process. No changes are required in any supervisor or kernel device driver processes. A number of file system utility programs have been added to the standard UNIX file system utility programs. A number of these deal with the file system inodes. One program was written specifically for the MERT 16-bit file system to minimize file system fragmentation. When a file system becomes congested, it must be unmounted and reconfigured by moving all files into contiguous areas, freeing up large numbers of contiguous free blocks. On a file system which is normally about one-half full, running the *recon* program is only required once a week or so. The equivalent of all of the standard UNIX file system utility programs have been written for the MERT file systems. In particular, the *dump* and *restor* programs have been generalized to make it easy to dump any file system format and restore it from tape onto a different file system format.

The original MERT 16-bit file system has received heavy usage in many different installations. It serves the purpose for which it was designed. The UNIX 16-bit file system has been used in the MERT system to enable the running of either the UNIX time-sharing system or the MERT real-time system on the same disk pack. The new MERT 32-bit file system was recently installed by W. A. Burnette on a running system and is essentially debugged at this time. It will be an easy matter to make this file system completely compatible with the UNIX 32-bit file system provided no contiguous files exist in the file system. The inodes are the same size (64 bytes) on both systems and the superblocks differ only in a few fields. Compatibility can be achieved by allocating a dummy file in the UNIX file system to include the bit map which is used by the MERT file system.

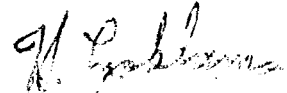
The MERT file system can make a significant improvement in the execution of program files by making the files contiguous using the *fmove* primitive (especially in the /bin and /usr/bin directories). Some potential problems still exist in the new file system. The larger size inode in memory means that fewer inodes can be stored in memory. The allocation strategy for blocks for time-sharing files means that the free list could be exhausted. When free blocks are returned, they are put on the free chain, not back in the bit map where they can be used for allocation to contiguous files. These blocks can only be reclaimed by a *check -s* salvage operation. This strategy does not provide for a free exchange between contiguous space and space described by the free chain.



The MERT 32-bit file system is not very flexible for dealing with files which are allocated by extents. Only 6 extents are available leading to a potential file growing problem. However all 6 extents are in memory when the file is open; thus only one disk access is required to read an arbitrary block, even if the file is allocated in pieces. This was not true for the MERT 16-bit file system. It is felt that a reasonable trade-off was made to make the file system more suitable for time-sharing applications and yet provide real-time capabilities where necessary.

**Acknowledgements**

The original 16-bit MERT file system was designed for the MERT system which was developed in collaboration with D. L. Bayer. The UNIX 16-bit file system was implemented on the MERT system by T. M. Raleigh by writing a MERT file manager process. The new MERT 32-bit file system was debugged and made into a running system by W. A. Burnette. All versions of the MERT file systems have benefited from the efforts of T. M. Raleigh and W. A. Burnette. E. A. Loikits provided many of the dump/restor conversion programs for the various file system formats.



**H. Lycklama**

HO-3114-HL-hl

*References*

- [1] K. Thompson and D. M. Ritchie, The UNIX Time-Sharing System, Comm. ACM 17, (July 1974), p365.
- [2] J. F. Maranzano, A Description of the UNIX File System, MF-75-8234-32.
- [3] H. Lycklama and D. L. Bayer, The MERT Operating System, TM-78-3114-3.
- [4] K. Thompson, private communication.
- [5] H. Lycklama, The MERT/UNIX Supervisor, TM-78-3114-4.
- [6] H. Lycklama and D. L. Bayer, The MERT Programmer's Manual, October 1977.

APPENDIX A - UNIX 16-bit File System Superblock

```
#define N_FREEL 100 /* number of free list extents */
#define N_FINODE 100 /* maximum number of free inodes */
/*
 * Definition of the unix super block.
 * The root super block is allocated and
 * read in iinit/alloc.c. Subsequently
 * a super block is allocated and read
 * with each mount (smount/sys3.c) and
 * released with unmount (sumount/sys3.c).
 * A disk block is ripped off for storage.
 * See alloc.c for general alloc/free
 * routines for free list and I list.
 */
struct    filsys
{
    unsigned s_ise; /* size in blocks of I list */
    unsigned s_fsize; /* size in blocks of entire volume */
    int s_nfree; /* # of in core free blocks (0-100) */
    daddr_t s_free[N_FREEL]; /* in core free blocks */
    int s_ninode; /* number of in core I nodes (0-100) */
    ino_t s_inode[N_FINODE]; /* in core free I nodes */
    char s_flock; /* lock during free list manipulation */
    char s_ilock; /* lock during I list manipulation */
    char s_fmod; /* super block modified flag */
    char s_ronly; /* mounted read-only flag */
    long s_time; /* current date of last update */
    int pad[40];
    int s_tfree; /* total # of free blocks */
    int s_tinode; /* total # of free inodes */
    char s_fname[6]; /* pack name */
    char s_fpack[6]; /* pack label */
};
```

APPENDIX B - UNIX 16-bit File System On-Disk Inode

```
/*
 * Inode structure as it appears on
 * the disk. Not used by the system,
 * but by things like check, df, dump.
 */
struct      vtoc
{
    int      v_mode;
    char     v_nlink;
    char     v_uid;
    char     v_gid;
    char     v_size0;
    char     *v_size1;
    daddr_t  v_addr[8];
    time_t   v_atime;
    time_t   v_mtime;
};

/* modes */
#define IALLOC      0100000      /* file is used */
#define IFMT        060000      /* type of file */
#define IFDIR      040000      /* directory */
#define IFCHR      020000      /* character special */
#define IFBLK      060000      /* block special, 0 is regular */
#define IFREC      070000      /* record special file */
#define ILARG      010000      /* large addressing algorithm */
#define ISUID      04000      /* set user id on execution */
#define ISGID      02000      /* set group id on execution */
#define ISVTX      01000      /* save swapped text even after use */
#define IREAD      0400      /* read, write, execute permissions */
#define IWRITE     0200
```

APPENDIX C - UNIX 16-bit File System In-Core Inode

```
/*
 * The I node is the focus of all
 * file activity in unix. There is a unique
 * inode allocated for each active file,
 * each current directory, each mounted-on
 * file, text file, and the root. An inode is 'named'
 * by its dev/inumber pair. (iget/iget.c)
 * Data, from mode on, is read in
 * from permanent inode on volume.
 */
struct      inode{
    char      i_flag;
    char      i_count;      /* reference count */
    dev_t     i_dev;        /* device where inode resides */
    ino_t     i_number;     /* i number, 1-to-1 with device address */
    int       i_mode;
    char      i_nlink;     /* directory entries */
    char      i_uid;       /* owner */
    char      i_gid;       /* group of owner */
    char      i_size0;     /* most significant of size */
    char      *i_size1;    /* least sig */
    daddr_t   i_addr[8];   /* device addresses constituting file */
    char      i_use;
};

/* flags */
#define ILOCK      01      /* inode is locked */
#define IUPD      02      /* inode has been modified */
#define IACC      04      /* inode access time to be updated */
#define IMOUNT    010    /* inode is mounted on */
#define IWANT     020    /* some process waiting on lock */
#define ITRUNC    040
#define ICRIT     0100
#define IMOVE     0200

/* modes */
#define IALLOC    0100000  /* file is used */
#define IFMT      060000   /* type of file */
#define IFDIR     040000   /* directory */
#define IFCHR     020000   /* character special */
#define IFBLK     060000   /* block special, 0 is regular */
#define IFREC     070000
#define ILARG     010000   /* large addressing algorithm */
#define ISUID     04000    /* set user id on execution */
#define ISGID     02000    /* set group id on execution */
#define ISVTX     01000    /* save swapped text even after use */
#define IREAD     0400     /* read, write, execute permissions */
#define IWRITE    0200
#define IEXEC     0100
extern struct inode inode[];
```

APPENDIX D - MERT 16-bit File System Superblock

```
#define N_FREEL      64      /* number of free list extents */
#define N_FINODE    60      /* maximum number of free inodes */
#define N_UPDATE    30      /* maximum number of update entries */

struct    filsys
{
    char    *s_ysize;      /* number of blocks of inodes */
    char    *s_fsize;     /* number of blocks in file system */
    struct  {
        daddr_t    stblk;
        unsigned    ncbkls;
    } s_ext[N_FREEL];
    int     s_ninode;     /* number of free inodes */
    int     s_inode[N_FINODE];
    int     s_nupdate;   /* number of update entries filled */
    struct  {
        daddr_t    stblk;
        unsigned    nublks;
    } s_update[N_UPDATE];
    char    s_flock;
    char    s_ilock;
    char    s_fmod;
    char    s_ronly;
    time_t  s_time;
};
```

APPENDIX E - MERT 16-bit File System On-Disk Inode

```
#define N_VEXT 27      /* number of extents in VTOC entry */

/*
 * Inode format on disk
 */
struct      vtoc {
    int      v_mode;          /* file mode bits          */
    char     v_nlink;        /* number of links to inode */
    char     v_uid;          /* uid owner                */
    char     v_gid;          /* gid owner                */
    char     v_invoc;        /* invocation count (uniqueness) */
    char     *v_size[2];     /* byte size                */
    struct   {
        daddr_t  stblk;      /* # start block #        */
        unsigned ncbks;     /* # of contiguous blocks */
    } v_ext[N_VEXT];        /* extents                */
    time_t   v_atime;        /* access time            */
    time_t   v_mtime;        /* modification time      */
    int      v_chksm;        /* checksum (not used)    */
};

/*
 * modes
 */
#define IALLOC 0100000 /* inode allocated */
#define IFMT 070000 /* file format */
#define IFDIR 040000 /* directory */
#define IFCHR 020000 /* character device */
#define IFBLK 060000 /* block device */
#define IFREC 070000 /* record device */
#define ICONT 010000 /* contiguous file */
#define ISUID 04000 /* set user id */
#define ISGID 02000 /* set group id */
#define IREAD 0400 /* read */
#define IWRITE 0200 /* write */
#define IEXEC 0100 /* execute */
```

APPENDIX F - MERT 16-bit File System In-Core Inode

```
/*
 * MERT incore inode format
 */
struct      inode{
    char      i_flag;                /* flags (see below) */
    char      i_count;              /* number of opens */
    dev_t     i_dev;                /* device on which file exists*/
    ino_t     i_number;             /* file node number */
    int       i_mode;               /* file mode bits */
    char      i_nlink;              /* number of links to inode */
    char      i_uid;                /* uid owner */
    char      i_gid;                /* gid owner */
    char      i_invoc;              /* invocation count (uniqueness) */
    char      *i_size[2];           /* byte size */
    int       i_extent[2];          /* current extent */
    int       i_stblk;              /* block number of first block of extent */
    char      i_extno;              /* extent number in core */
    char      i_use;                /* inode use count */
};

/*
 * flags
 */
#define ILOCK      01      /* inode locked */
#define IUPD      02      /* update modification time*/
#define IACC      04      /* update access time */
#define IMOUNT    010     /* inode is a mount point */
#define IWANT     020     /* inode wanted */
#define ITRUNC    040     /* truncate inode to new size*/
#define ICRIT    0100     /* critical region */
#define IMOVE     0200     /* inode being moved */

/*
 * modes
 */
#define IALLOC    0100000   /* inode allocated */
#define IFMT      070000    /* file format */
#define IFDIR     040000    /* directory */
#define IFCHR     020000    /* character device */
#define IFBLK     060000    /* block device */
#define IFREC     070000    /* record device */
#define ICONT     010000    /* contiguous file */
#define ISUID     04000     /* set user id */
#define ISGID     02000     /* set group id */
#define IREAD     0400     /* read */
#define IWRITE    0200     /* write */
#define IEXEC     0100     /* execute */
```



APPENDIX G - UNIX 32-bit File System Superblock

```
/*
 * Definition of the unix super block.
 * The root super block is allocated and
 * read in iinit/alloc.c. Subsequently
 * a super block is allocated and read
 * with each mount (smount/sys3.c) and
 * released with unmount (sumount/sys3.c).
 * A disk block is ripped off for storage.
 * See alloc.c for general alloc/free
 * routines for free list and I list.
 */
struct    filsys
{
    unsigned short s_ysize;    /* size in blocks of I list */
    daddr_t      s_fsize;    /* size in blocks of entire volume */
    short        s_nfree;    /* number of in core free blocks */
    daddr_t      s_free[NICFREE]; /* in core free blocks */
    short        s_ninode;    /* number of in core I nodes */
    ino_t        s_inode[NICINOD]; /* in core free I nodes */
    char         s_flock;    /* lock during free list manipulation */
    char         s_iloc;    /* lock during I list manipulation */
    char         s_fmod;    /* super block modified flag */
    char         s_ronly;    /* mounted read-only flag */
    time_t       s_time;    /* current date of last update */
    short        s_dinfo[4]; /* device information */
    daddr_t      s_tfree;    /* Total free, for subsystem examination */
    ino_t        s_tinode;    /* Free inodes, for subsystem examination */
    char         s_fname[6]; /* File system name */
    char         s_fpack[6]; /* File system pack name */
};
```

*APPENDIX H - UNIX 32-bit File System On-Disk Inode*

```
/*
 * Inode structure as it appears on
 * a disk block.
 */
struct    dinode
{
    short    di_mode;
    short    di_nlink;
    short    di_uid;
    short    di_gid;
    off_t    di_size;
    char     di_addr[40];
    time_t   di_atime;
    time_t   di_mtime;
    time_t   di_ctime;
};

#define    INOPB    8        /* 8 inodes per block */

/*
 * the 40 address bytes:
 *     39 used; 13 addresses
 *     of 3 bytes each.
 */
```

APPENDIX I - UNIX 32-bit File System In-Core Inode

```
/*
 * The I node is the focus of all
 * file activity in unix. There is a unique
 * inode allocated for each active file,
 * each current directory, each mounted-on
 * file, text file, and the root. An inode is 'named'
 * by its dev/inumber pair. (iget/iget.c)
 * Data, from mode on, is read in
 * from permanent inode on volume.
 */

#define NADDR 13
struct inode
{
    char i_flag;
    char i_count; /* reference count */
    dev_t i_dev; /* device where inode resides */
    ino_t i_number; /* i number, 1-to-1 with device address */
    short i_mode;
    short i_nlink; /* directory entries */
    short i_uid; /* owner */
    short i_gid; /* group of owner */
    off_t i_size; /* size of file */
    union {
        struct {
            daddr_t i_addr[NADDR]; /* if normal file/dir
            daddr_t i_lastr; /* last logical block read */
            struct {
                daddr_t i_rdev; /* i_addr[0] */
            };
        } i_un;
    };
};

extern struct inode inode[]; /* The inode table itself */

/* flags */
#define ILOCK 01 /* inode is locked */
#define IUPD 02 /* inode has been modified */
#define IACC 04 /* inode access time to be updated */
#define IMOUNT 010 /* inode is mounted on */
#define IWANT 020 /* some process waiting on lock */
#define ITEXT 040 /* inode is pure text prototype */
#define ICRT 0100 /* inode has been created */

/* modes */
#define IFMT 0160000 /* type of file */
#define IFDIR 0040000 /* directory */
#define IFCHR 0020000 /* character special */
#define IFBLK 0060000 /* block special */
#define IFREG 0100000 /* regular */
#define ISUID 04000 /* set user id on execution */
#define ISGID 02000 /* set group id on execution */
#define ISVTX 01000 /* save swapped text even after use */
```

```
#define IREAD 0400
#define IWRITE 0200
#define IEXEC 0100
```

```
/* read, write, execute permissions */
```

APPENDIX J - MERT 32-bit File System Superblock

```
/*
 * Definition of the mert super block.
 * The root super block is allocated and
 * read in initfsys/task.c. Subsequently
 * a super block is allocated and read
 * with each mount (smount/fmgr3.c) and
 * released with unmount (sumount/fmgr3.c).
 * A disk block is ripped off for storage.
 * See alloc.c and bmap.c for general alloc/free
 * routines for free list and I list.
 */

#define NICINOD 100 /* number of incore inodes */
#define NICFREE 50 /* number of incore free blocks */

struct filsys
{
    unsigned s_ysize; /* size in blocks of I list */
    daddr_t s_fsize; /* size in blocks of entire volume */
    unsigned s_nfree; /* number of in core free blocks */
    daddr_t s_free[NICFREE]; /* in core free blocks */
    int s_ninode; /* number of in core I nodes */
    ino_t s_inode[NICINOD]; /* in core free I nodes */
    char s_flock; /* lock during free list manipulation */
    char s_ilock; /* lock during I list manipulation */
    char s_fmod; /* super block modified flag */
    char s_ronly; /* mounted read-only flag */
    time_t s_time; /* current date of last update */
    daddr_t s_tfree; /* Total free, for subsystem examination */
    ino_t s_tinode; /* Free inodes, for subsystem examination */
    int s_m; /* interleave factor */
    int s_n; /* " */
    char s_fname[6]; /* File system name */
    char s_fpack[6]; /* File system pack name */
    daddr_t s_cfree; /* Free blocks on chain */
    daddr_t s_nxtblk; /* Next free block not on chain */
    daddr_t s_nxtcon; /* Next available contiguous area */
};

#define NHWAT 500
#define NLWAT 50
```

*APPENDIX K - MERT 32-bit File System On-Disk Inode*

```
/*
 * Inode structure as it appears on
 * a disk block.
 */
struct    dinode
{
    int        di_mode;
    int        di_nlink;
    int        di_uid;
    int        di_gid;
    off_t      di_size;
    char       di_addr[40];
    time_t     di_atime;
    time_t     di_mtime;
    time_t     di_ctime;
};

#define      INOPB      8          /* 8 inodes per block */

/*
 * the 40 address bytes:
 *      39 used; 13 addresses
 *      of 3 bytes each.
 */
```

APPENDIX L - MERT 32-bit File System In-Core Inode

```
/*
 * The I node is the focus of all
 * file activity in mert. There is a unique
 * inode allocated for each active file,
 * each current directory, each mounted-on
 * file and the root. An inode is 'named'
 * by its dev/inumber pair. (iget/iget.c)
 * Data, from mode on, is read in
 * from permanent inode on volume.
 */

#define NADDR 13
#define N_VEXT 6
struct inode
{
    char i_flag;
    char i_count; /* reference count */
    dev_t i_dev; /* device where inode resides */
    ino_t i_number; /* i number, 1-to-1 with device address */
    int i_mode;
    int i_nlink; /* directory entries */
    int i_uid; /* owner */
    int i_gid; /* group of owner */
    off_t i_size; /* size of file */
    daddr_t i_addr[NADDR]; /* if normal file/directory */
    char i_invoc; /* invocation count */
    char i_use; /* inode usage count */
};

/*
 * struct def'n for files allocated by extents
 */

struct i_extent {
    daddr_t stblk;
    daddr_t ncbkls;
};

/* flags */
#define ILOCK 01 /* inode is locked */
#define IUPD 02 /* inode has been modified */
#define IACC 04 /* inode access time to be updated */
#define IMOUNT 010 /* inode is mounted on */
#define IWANT 020 /* some process waiting on lock */
#define ITRUNC 040 /* inode to be trunc. to proper length */
#define ICRT 0100 /* inode has been created */
#define ICRT 0200 /* critical section */

/* modes */
#define IFMT 0170000 /* type of file */
#define IFDIR 0040000 /* directory */
#define IFCHR 0020000 /* character special */
#define IFBLK 0060000 /* block special */
#define IFREG 0100000 /* regular */
```

```
#define IFREC 0160000 /* record */
#define IFEXT 0120000 /* allocated by multiple contiguous extents */
#define IF1XT 0130000 /* contiguous file of 1 extent */
#define ISUID 04000 /* set user id on execution */
#define ISGID 02000 /* set group id on execution */
#define IREAD 0400 /* read, write, execute permissions */
#define IWRITE 0200
#define IEXEC 0100
```