Published: 12/13/66
(Supersedes: BG.0, 5/2/66)

Identification

Overview of the Basic File System
R.C. Daley, P.G. Neumann, D.M. Ritchie

Purpose

Section BG.0 presents an overview of the program structure of the basic file system. It does not pretend to be complete, since that is the task of Sections BG.1 through BG.17, which are detailed specifications of the individual modules and data bases.

References

A summary of the major goals of the Multics file system is contained in section 2 of "A general-purpose file system for secondary storage," by R. C. Daley and P. G. Neumann, Fall Joint Computer Conference, 1965. Although much of that paper has been superseded by sections of this manual, it is still useful for background information and motivation. The detailed commands to the basic file system and a summary of how the file system looks to the user may be found in Section BX.8 of this manual. It is recommended that Section BX.8 be read before the remaining BG sections. It is suggested that Section BG.0 be consulted whenever confusion arises in Sections BG.1 to BG.17 due to questions of gross flow of control.

Introduction

The Multics file system consists of two parts, the basic file system, and the backup and multilevel storage management system. The basic file system is that part of the hardcore supervisor which manages segments. It includes the transportation of pages of segments into and out of core from and onto on-line secondary storage. It also includes a hierarchical organization of segments into directories, and a means of controlling the way in which these segments may be used. The backup system, on the other hand, is that part of Multics which moves infrequently used segments downward to slower speed on-line devices and provides copies of all segments on off-line devices for retrieval and protection against catastrophe. These systems provide the user with a completely device-independent secondary storage system whose care and feeding is done without his cognizance. Section BG of this manual specifies in detail the design of the basic file system, while Section BH specifies the design of the backup system.

In the Multics system, a segment is simply a linear array of data which is referenced by means of a symbolic name (or segment number) and a linear index. In general, a user will not know, or need to know, how or on what device a segment is stored, as this is the responsibility of the file system.

A user may reference data within a segment either explicitly or implicitly. In the latter case, the user merely refers to his data during the normal execution of his process and the data are automatically provided by the file system. This implicit form of access to a segment is referred to as segment addressing and is accomplished by the file system in conjunction with the paging and segmentation hardware.

A segment may also be referenced explicitly throught the use of traditional read and write statements. This explicit form of access to a segment is called file addressing and is accomplished by means of formal calls to the Multics I/O system.

Input or output requests which are directed to I/O devices other than segments (i.e., tapes, teletypes, printers, card readers, etc.) are processed directly by a Device Interface Module which is designed to handle I/O requests for that device. However, I/O requests which are directed to a segment are processed by a special procedure known as the File System Interface Module. This module acts as a device interface module for segments within the file system. Unlike other device interface modules, this procedure does not explicitly issue I/O requests directly to the device. Instead, the file system interface module accomplishes its I/O implicitly by means of segment addressing. See Section BF for details regarding the I/O system.

Whether a user refers to a segment through the use of read and write statements or by means of segment addressing, ultimately a segment must be made available to his process by the basic file system. In general the basic file system performs the following functions.

1. Maintains directories of existing segments.

2. Makes segments available to a process upon request.

3. Creates, truncates and deletes segments.

Figure 1 is a rough block diagram of the modules and data bases which make up the basic file system. As such, it provides a census for Section BG of this Manual. The solid lines indicate the flow of control through the use of formal calling sequences (with calls in the direction of the arrows, where formal returns are implied). The circles in the diagram indicate some of the data bases within the basic file system. Dashed lines indicate the flow of data between modules and data bases. The modules and data bases drawn below the dotted line must reside in core memory at all times since they are invoked during a missing-page fault.

Normally, all of the modules of the basic file system are run as a part of a user's process whenever that process requires any part of a segment. These modules are segment control, page control, core control, directory control, access control, and the device interface modules. Segment control is called to establish a segment number, and to activate, truncate, reassign or terminate segments known to the given process. Each segment known to a process is described in a table for that process called the known segment table (KST). Directory control is called by the user to manipulate directory entries, e.g., to rename or delete a segment or to alter its access rights. These two modules are the only ones which may be called by user procedures. In addition, page control is invoked as a result of a missing-page fault, making use of certain paging information kept in the system segment tables (SST). Similarly, segment control is invoked as a result of a segment fault (e.g., segment not known, segment inactive, etc.). The remaining modules are completely internal to the basic file system. Core control is the module which accounts for the availability and allocation of core memory, information about which is maintained in the core map. Access control determines the access rights for the given user in using a particular segment. The device interface modules (DIMs) are the modules which know about the peculiarities of the secondary storage devices, and which actually control input/output; There is a drum DIM for the high-speed drum (which has its own controller), and a disc DIM for the DS25 disc (which uses the General Input-Output Controller, or GIOC).

Segment Control

Segment control maintains records of all segments known to the current process. By definition a segment is known to a process if it has been assigned a segment number by the process. A table of such segments is maintained by segment control; this is the known segment table (KST). Each process has its own KST, containing, for each entry, a unique identifier which unambiguously identifies the segment among all that exist or have existed in the system. The KST entry for a directory segment also contains a list of symbolic names by which the process may refer to the segment. These names are used in communication with directory control. Finally, a KST entry contains other information pertinent to the use of this segment by this process, such as the segment number of the segment and access control information.

If a segment is known to some process, it may also have an entry in a set of tables called the _system segment tables_ (SST). The SST is maintained on a system-wide basis; at most one entry appears for a single segment, even though it may be known to many processes. The SST consists of the _active segment table_ (AST), the _descriptor segment table_ (DST), and the _process segment table_ (PST).

A non-descriptor segment known to some process may have an entry in the AST. In this case the segment is called _active_, and from information in the AST entry a page table may be created or a missing page restored. Each AST entry has at least one _active file trailer_ (AFT), giving information on the location in secondary storage of the file corresponding to the segment. One file per segment is the normal case. However, the backup and multilevel system may request that a file be moved from one device to another. While this movement is in progress, there are two AFT trailers for the AST entry, one for the original file and one for the _move_ file. Moving is done dynamically as a result of the normal transportation of pages in and out of core; when moving is complete the move file will replace the original file in the hierarchy. A segment for which an AST entry exists is, by definition, _active_; if in addition its page table is in core, the segment is called _loaded_. Conceptually, a loaded segment need have no pages in core; the term "loaded" refers only to the page table. (In fact, page control will usually dispose of the page table when the number of pages in core drops to zero.)

The table for descriptor segments corresponding to the AST is the descriptor segment table (DST). There is an entry in this table for each descriptor segment which is loaded, that is, for which the page table is in core. When the page table of a descriptor segment is removed, the descriptor segment is destroyed, rather than moved out to secondary storage. Thus, the concept of active does not apply to descriptor segments.

Each active process has an entry in the PST whose function is to summarize information about certain important segments in the process. An active process may be taken to be a process which has an entry in the PST and such that the segments mentioned in the PST are active segments. (See also section BJ.2.01) Inactive processes do not have entries in any of the wired-down data bases in the system, although the traffic controller maintains a paged table of known processes. (See section BJ.2.02)

When a user wishes to make a segment known to his process, he calls directory control to find the branch of the segment in the appropriate directory. When the branch is found, directory control calls segment control to establish the segment as known to the process. Segment control takes the following steps in making a segment known:

1. Assign a segment number to this segment.

2. Create a new entry in the KST indicating that this segment is now known to the given process.

3. Determine the descriptor control bits for the segment from the effective mode of the segment branch, and save them in the KST.

4. Return the segment number to the calling procedure.

When a segment first becomes known to a process, it may be inactive. (It could also be active, because another process might be using the segment as well.) When a reference is made to an inactive segment, a segment fault occurs. Segment control is called, and the following steps are taken to activate the segment.

5. Using the segment number, locate this segment in the KST, and obtain the unique identifier for the segment.

6a. Search the AST for a segment with this unique identifier. If none is found, create an AST entry by calling directory control to get the needed information from the directory branch pointing to the segment. (The KST has a pointer to the proper directory.)

6b. Whether the AST entry already existed or had to be created, enter this process in a part of the AST entry containing a list of processes currently using this segment.

7. Establish the segment descriptor in the descriptor segment.

8. Read in the referenced page or pages via a call to page control; page control will establish the page table (load the segment) if necessary.

As a result of steps 5. through 8., the segment is now active and loaded.

In step 6a above, it was necessary to find the directory branch pointing to a segment in order to activate the segment. If this directory is inactive when directory control attempts to read it, another segment fault will occur and it will be necessary to activate the directory segment. This in turn may cause another segment fault when directory control attempts to read the directory superior to the directory; this sequence of faults will continue until an active directory is found, but must ultimately terminate because the root directory is always active.

In addition to the functions described above, segment control provides entries through which the user may ask questions or make declarations involving the use of segments known to his process. Some of these functions are listed below.

1. Declare that some specific locations within a segment are no longer needed at this time, and should be written onto secondary storage.

2. Declare that some specific locations within a segment are to be required shortly.

3. Terminate a segment, indicating that this segment is no longer to be considered as known to the calling process.

4. Truncate a segment.

Directory Control

Directory control provides all the basic tools for manipulating entries within a directory. It may be called by segment control or directly as a result of a file system command. The functions provided by this module perform primitive operations and are usually augmented by more elaborate system library procedures (see BX.8). The following is a list of some of these operations.

1. Create a new directory entry.

2. Delete an existing entry.

3. Rename an entry.

4. Return status information concerning a particular entry or entries.

5. Change the access control information for a particular branch.

6. Find a branch and make the segment to which it points known to the process.

Whenever a user wishes to perform any operation on the contents of a segment pointed to by a particular directory entry, the branch pointing directly to that segment is first obtained from that directory. Access control is then called to determine what permission the user may be granted.

Thus, when directory control is called from outside the basic file system (e.g., "call status(_pathname_, _entryname_,

...);"), directory control must be able to read the necessary directory (i.e., that specified by pathname) as a segment. To do this directory control asks segment control to search the KST for a directory segment with the proper name. If there is such an entry in the KST, then segment control returns the corresponding segment number. If, however, the directory is not known, segment control calls directory control in order to get the information needed to establish the directory in the KST: namely, the information in the branch pointing to the directory; this branch is contained in the next superior directory. In order to read this superior directory, directory control must recursively call segment control to obtain a segment number for it. This recursion terminates when a known directory is found or when the root, which is known to directory control, is reached.

Access Control

Access control is called by directory control to evaluate the access control information for a particular branch. Thus the name of the user and the access control information for the branch in question is made available to access control. The access control returns a single effective mode to directory control. The effective mode is the mode which governs the use of a file with respect to the current user or process. This effective mode is used by directory control to determine if the requested operation is to be permitted.

The apparent mode of a branch (specified in the access control information) consists of the TRAP, READ, EXECUTE, WRITE and APPEND attributes, defined in Section BX.8. If the TRAP attribute is OFF for a given user, then the effective mode is in fact the apparent mode. If, however, the TRAP attribute is ON, the effective mode is obtained as follows. Associated with the TRAP attribute is the path name of a single procedure to be called, along with a list of parameters to be passed to this procedure (unless the user has inhibited traps, in which case an error return indicates that access has been denied). This procedure is also passed the user's name, his apparent mode and the names of the directory and branch in question. This procedure must return the effective mode to the access control module. As a result, this procedure may alter the user's apparent mode of the branch in question. The TRAP attribute is extremely useful for monitoring of segment usage, for applying special restrictions on segment usage not within the framework of the usage attributes (e.g., lock mechanisms), for avoiding the necessity of creating a segment until it is actually needed (e.g., something which might never be needed), for retrieving a segment no longer residing on the on-line storage system (see Section BH), etc.

There are facilities for system traps of a similar nature which are however not available to normal users. These system traps are not inhibitable, and are also not detectable. They are present so that the system may gather information for its own use.

Page Control

Page control is responsible for overseeing all segment I/O and for maintaining page tables. For example, page control may be called by segment control to perform input or output operations for pages of segments. When a process references a word in an inactive segment, segment control, after activating the segment, calls page control to obtain the page containing the word requested. If the segment is not already loaded, page control first creates a page table, each word of which contains a fault indication and a flag specifying what should be done when and if that fault occurs. The flag may indicate either that a blank page should be assigned or that the referenced page should be read in from secondary storage. Then the referenced page is read in if necessary and its page table word is set to point to the proper core location.

Control also passes to page control by means of a missing-page fault in a page table in use by the current process. This fault may indicate that a new page should be assigned or that an existing page should be read in from an active file. In either case a block of core must be assigned before anything else can happen. This is accomplished by a call to core control.

If a new page is to be read in, the page table entry for the missing page contains a pointer to the appropriate entry in the system segment tables. If the segment is being moved from one device to another, there is a flag indicating whether this page has already been moved. If it has been, it is read from the new device. In any case the AFT trailer on the AST entry for the segment contains a pointer to the appropriate location in secondary storage. This pointer is passed as a parameter to the DIM with a read request to bring the correct page to core memory.

Finally, core control may call page control to indicate that a page should be removed because of low usage. If the segment containing the page is being moved, the page is always written out onto the new device and its page table word is marked so that the move is remembered.

When the last page of a segment is removed from core, page control usually releases its page table. If the segment is being moved, page control checks the page table words to find if there are any pages that haved not been moved. If there are, one of the unmoved pages is read into core. This page is soon removed because no one uses it, and the next unmoved page is read in. This sequence continues until the move is complete when the segment will at last be unloaded.

Core Control

Core control provides core space for pages and page tables to page control. It maintains information on the availability of each block of core and how this block is being used. This information is kept in the <u>core map</u>, and includes the availability of the block to the system, whether the contents of the block may be removed from core, and their eligibility for removal.

Those blocks whose contents are candidates for removal are indicated in the core map by a usage list in the order of decreasing frequency of usage. This is a linked list in the core map. At periodic intervals, the list is reordered by core control on the basis of use in that period. When blocks of core are required for assignment, it is in general necessary to remove some pages from core. These pages are chosen from the bottom of the usage list, and are removed via page control. In the actual implementation, a pool of unassigned core blocks is maintained so that requests may in fact normally be satisfied simply and quickly, without the prior removal of pages from core. This supply of unassigned core is replenished by the removal of pages as required.

Device Interface Modules

For each type of on-line secondary storage device used by the basic file system, a device interface module is provided. A device interface module has the sole responsibility for the strategy to be used in dealing with the particular device for which it was written. Any special considerations pertaining to a particular storage device are invisible to all modules except the interface module for that device.

A device interface module is also responsible for assigning physical storage areas, as needed, on the device for which it was written. To accomplish this function, the interface module must maintain records of all storage already assigned on that device. These records are kept in <u>file maps</u> which reside on the device to which they refer. The DIMs also maintain <u>free storage maps</u> to aid in assigning locations for new information sent to their devices.

Each DIM accepts four types of requests:

1. Read

2. Write

3. Truncate

4. Clean up

Read and write requests specify the core location from or to which the affected information is to be moved, the location of this information within its segment, and a pointer to an AFT that gives the location, on the DIM's device, of the file map for the segment that is involved. The DIM reads the file map for the segment to find the location of the affected pages and then performs the indicated I/O. Portions of the file maps for segments with a high I/O rate are kept in core in a _DIM history_ table to avoid the extra read operations necessary to get the file map.

A truncate request specifies an AFT and a 64 word record beyond which the segment is to be deleted. The file map is read and the records beyond the limit are placed on the free storage map. Truncation from record 0 causes the entire segment and its file map to be released.

The cleanup request is given to inform the DIM that the specified AFT is about to be removed and that the DIM should remove all core-resident information for this segment from the DIM history table.

Each request given to a DIM is placed in a queue. Since several I/O operations may be needed to complete each request, the DIM does not attempt to finish a request before going on to the next, but instead will do as much work as possible on each request and then go on to the next request in the queue. Each time a DIM is called, whether with a new request or as a result of an I/O interrupt, the queue is scanned to see if there is any work that can be done. Whenever a request is finished, page control is called and informed of this fact.

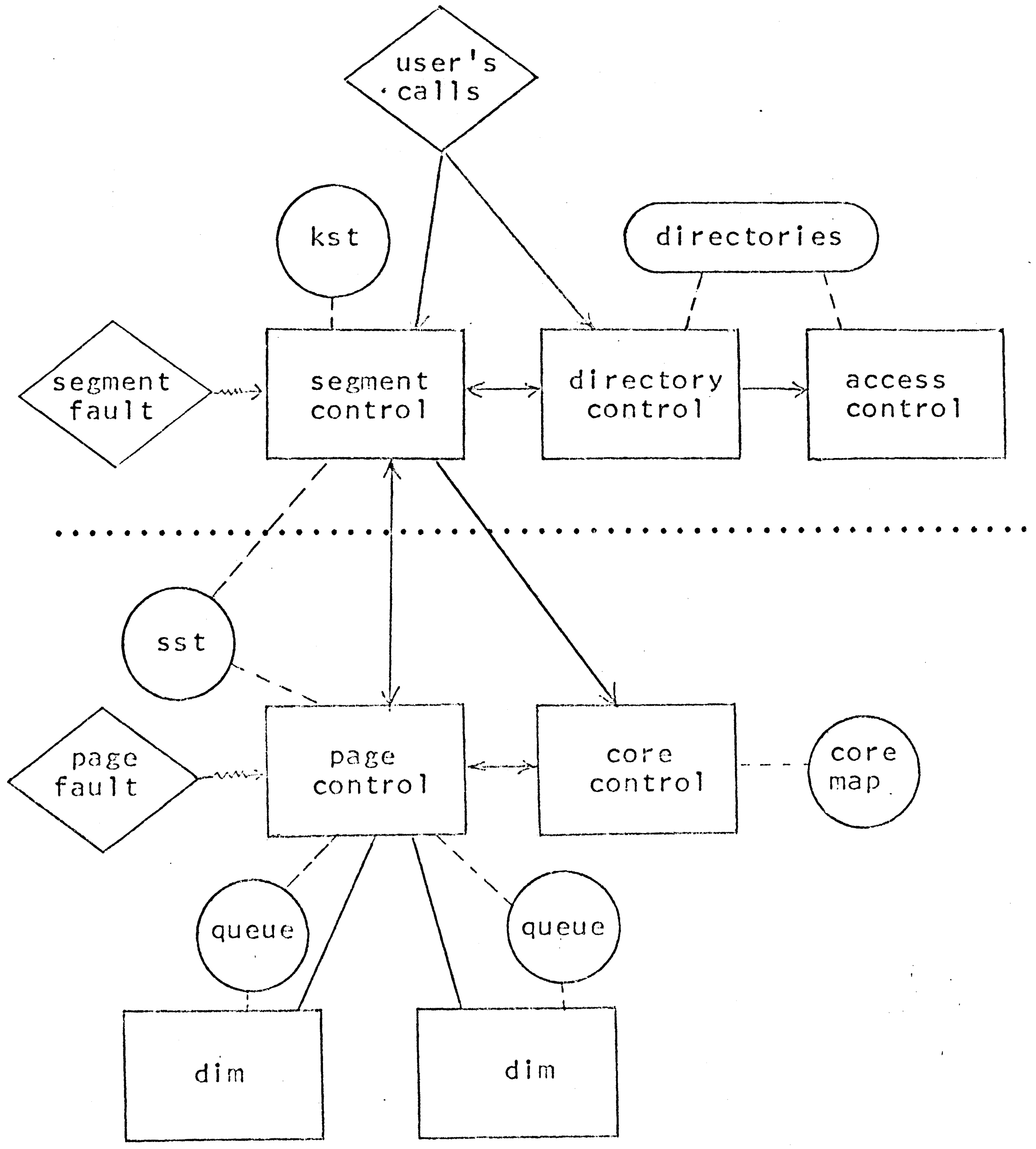

Figure 1. The Basic File System