TO: MSPM Distribution
FROM: P. G. Neumann
DATE: 05/15/68
SUBJECT: I/O Switch Complex

Attached are BF.2.10, 11, 12 and 13, completing the documentation on the I/O Switch Complex, complementing BF.2.14 and 15, published 2/28/68. BF.2.10 supersedes the earlier version of 03/01/67, and is a minor modification.

Published: 05/15/68
(Supersedes: BF.2.10, 03/01/67)

Identification

Overview of the Switching Complex
D. A. Levinson

Purpose

The switching complex (see Figure 2) plays a central role in threading control through the modules of the I/O system involved in the processing of a particular stream, frame or item.

The contents of this document are as follows. First, the role of the switch is explained in terms of the concept of a software path. Then follows a summary of the modules and data bases of the switching complex. After the summary, brief operational descriptions of the switching complex are given. Finally, some of the considerations which led to the current design are explained.

The Software Path

The I/O system supports a number of devices and offers a number of logical services from which the user may choose. Internally, these devices and services are represented as modules, called outer modules. For example, to each device supported by the I/O system corresponds a device interface module (DIM). Every DIM is an outer module. To each of the services of logical record formatting and broadcasting corresponds a logical service module (LSM). Every LSM is an outer module. Structurally, each outer module is a segment with an entry point corresponding to each outer call (user call) which is meaningful for it.

Comparatively few outer modules are involved in the processing of the outer calls for a given ioname. What follows describes the selection of the outer modules for processing a given ioname and how the selected modules form a "path" or sequence of modules through which control flows when the user issues outer calls referencing the given ioname.

The user initiates processing of a stream, frame or item by issuing an attach call (see BF.1.01). The ioname (first) argument of the attach call serves as an identifier for the stream, frame or item. The type (second) and mode (fourth) arguments of the attach call specify a selection from the available devices and services which are to apply to the given stream, frame or item. After attachment, the user operates on the stream, frame or item by issuing _outer calls_ (standard user calls). The I/O system "knows" which stream, frame or item is being referenced by the ioname which is the first argument of every outer call.

Internally, the _type_ and _mode_ arguments of the attach call determine a software path, or sequence of modules, for processing outer calls which specify the same ioname as the attach call.

Formally, a _software path_ for processing a given outer call specifying a given ioname is defined as a sequence of pairs of modules such that,

1) the first module of each pair is a special module of the I/O system called the _I/O Switch_ and the second module is an outer module, and,

2) the sequence of pairs is _terminated_ by the file system interface module (FSIM) or _followed_ by the GIOC interface module (GIM).

A few remarks concerning the definition are vital to avoid misunderstanding. Foremost is the observation that the multiple appearances of the I/O Switch do not represent separate instances but recursive calls. Also, it is important to note that the path determined by the _type_ and _mode_ arguments of the attach call, is subject to variation. In fact, a given outer module may call on the "next" more than once, or not at all, depending on the outer call, its arguments and the state of the stream, frame or item at the time of the call. Furthermore, the path is subject to variations for the handling of errors and exceptions.

To distinguish the software path established by an attach call from the preceding variations, it is called the _basic software path_. Figure 1 is a diagram of the basic software path established by the following:

```
call attach("henry", "tape", "reel 432", "ws", status);
```

The circled crosses of Figure 1 represent appearances of the I/O Switch. As in the definition, the diagram represents recursive calls to the I/O Switch as separate appearances.

During processing of an attach call for an "unknown ioname" (an ioname not contained in the Attach Table) a special outer module, the Not Founder, temporarily appears as the first outer module of the path. Its function is to make an Attach Table entry for the "unknown ioname". By making the entry for the ioname, the Not Founder automatically removes itself from the path of future calls specifying the given ioname. While in the path, the Not Founder determines a "next" outer module on the basis of the type specified in the attach call. But, recall that the basic software path is a function of the mode argument of the attach call as well as the type. Hence, the "next" outer module may negotiate to have itself replaced by another module, or may cause another module to be "spliced" into the path before it, or after it, or both. Upon return to the user, the software path has settled down to the basic software path.

Once established, a representation of the basic software path is embedded in the Attach Table. A unique entry in the Attach Table corresponds to each appearance of the I/O Switch in the basic software path. Equivalently, there is one and only one entry per ioname. Each entry associates an ioname with a unique outer module in the path. Each outer module "knows" the ioname associated with the next outer module in the path.

When the user issues an outer call, the I/O Switch uses the specified ioname to reference the Attach Table. The I/O Switch forwards the call to the outer module associated with the ioname. Outer modules drive control along the software path by issuing calls, which specify the ioname associated with the next outer module in the path, to the I/O Switch.

Summary of Modules and Data Bases

The modules and data bases described in the following paragraphs are described in detail in other sections of the MSPM. References are made to these sections at appropriate points in the discussion.

In Figure 2, solid rectangles represent modules and circles represent data bases. Lines with arrows indicate flow of control by standard calls. Lines without arrows connecting data bases with modules, indicate referencing and/or updating of the data base by the module to which it is connected.

The two broken-line rectangles show the modules invoked in important cases. Detailed descriptions of these cases are given later. Here, it is worth noting that the larger rectangle is the more exceptional the case.

Entry Point Vectors

An Entry Point Vector (EPV) for a given outer module of a given process is a one-dimensional array of pointers (links) to the entry points of the given outer module. At process initiation no EPVs exist. They are constructed dynamically by the Entry Point Vector Maker (EPVMK). An outer module is linked if an EPV for it has been constructed. The I/O Switch (IOSW) uses EPVs to forward outer calls to outer modules.

For further details see BF.2.11 and BF.2.15.

The Type Table

To each type (valid value of the second argument of the attach call, see BF.1.01) corresponds an outer module. For each process the Type Table (TT) associates the type with the corresponding outer module. Thus, the TT is initialized with an entry for each standard type. Among other items, each entry of the TT contains:

1) the type mnemonic,

2) the name of the corresponding outer module.

All references to the TT are made through the Type Table Maintainer (TTM). The TT is a per process table which may be altered by the working process in which it resides (see the following paragraph).

The Type Table (TT) is updated by the module (see BF.1.09) which responds to the outer call (not yet defined) which allows the user to replace the module corresponding to a given type or to add a module and a corresponding new type.

When the Not Founder (NF) constructs an Attach Table (AT) entry for a newly attached ioname, it references the Type Table (TT) to obtain the pointer to the Entry Point Vector (EPV) corresponding to the type specified in the attach call.

For further details see section BF.2.14.

The Attach Table

For each user group, the Attach Table (AT) associates a given ioname with the outer module corresponding to the type specified in the attach call. There are provisions for per-process specialized correspondences. This will be elaborated in subsequent versions of this document. Thus, among other items, each entry of the AT contains:

1) the ioname,

2) a pointer to the EPV for the outer module corresponding to the type specified in the attach call.

All references to the AT are made through the Attach Table Maintainer (ATM).

The Not Founder (NF) updates the AT with entries for newly attached ionames. Other modules of the I/O system update the AT in order to establish the software path for processing calls referencing a given ioname. Various modules of the I/O system update existing entries of the AT with per ioname data for which the AT serves as the data base.

The I/O Switch (IOSW) references the Attach Table (AT) in order to obtain the pointer to the Entry Point Vector (EPV) for the outer module to which it forwards control. Various modules of the I/O system reference the AT to obtain data tabled with the ioname.

For further details see section BF.2.13.

The Attach Table Maintainer

The Attach Table Maintainer (ATM) services all requests to reference and/or update the Attach Table (AT).

The ATM performs a special action when an attempt is made to retrieve a pointer to an Entry Point Vector (EPV) associated with an ioname not contained in the AT. In this case, it returns a pointer to the EPV for the Not Founder (NF). Also, on the first such instance of the above, the ATM calls the Type Table Maintainer (TTM) in order to force linking of the NF.

For further details see section BF.2.13.

The I/O Switch

When the I/O Switch (IOSW) receives a call referencing a given ioname, it issues a call to the Attach Table Maintainer (ATM) to retrieve the pointer to the Entry Point Vector (EPV) for the outer module associated with the ioname. It then uses that pointer to obtain the link to the proper entry point of the outer module. The I/O Switch (IOSW) forwards the call to the entry point of the outer module to which the link points.

For further details see section BF.2.11.

The Not Founder

The Not Founder (NF) is the outer module called by the I/O Switch (IOSW) whenever the ioname (first) argument of an outer call is not found in the Attach Table (AT). When called at its attach entry point, its role is to make an entry for the given ioname in the AT. The NF references the Type Table (TT) through the Type Table Maintainer (TTM) in order to obtain the pointer to the Entry Point Vector (EPV) for the outer module corresponding to the type specified in the attach call. After updating the AT, it re-issues the attach call to the IOSW. This time the ATM returns the pointer to the EPV for the appropriate outer module (not the pointer to the EPV for the NF).

For further details see section BF.2.12.

The Type Table Maintainer

The Type Table Maintainer (TTM) services all requests to reference and/or update the Type Table (TT).

The TTM performs a special action when a type is referenced for which the pointer to the Entry Point Vector (EPV) is null. In this case, it calls the Entry Point Vector Maker (EPVMK) to construct the EPV for the outer module corresponding to the type. EPVMK returns a pointer to the EPV. The TTM replaces the original null pointer in the entry for the type with the returned pointer.

For further details see section BF.2.14.

The Entry Point Vector Maker

The Entry Point Vector Maker (EPVMK) constructs the Entry Point Vector (EPV) for a given outer module. It is called by the Type Table Maintainer (TTM) when the TTM receives a request for a type entry for which the Entry Point Vector pointer item is null. It has a table (not shown in Figure 2) of all entry point names for outer modules and the outer module name is passed to it by argument. It therefore, has all the information needed to call the supervisory routines LINK and LINKMK in order to obtain forced links to the entry points of the outer module. It allocates the required storage, stores the links as a one-dimensional array and returns a pointer to it by argument.

For further details see section BF.2.15.

Operational Descriptions

Descriptions are given for the two important cases corresponding to the nest of broken-lines in Figure 2. Each description is keyed to a separate figure as noted in the headings. These figures have the same drawing conventions as lines are sequence numbers for the calls. Both the figures and the descriptions lean heavily on the previous sections. Accordingly, the abbreviations for the modules and data bases are used, and the descriptions of the actions are brief. In the descriptions, each "paragraph" begins with the abbreviation of the module which is performing the action described. Changes in control from module to module by call or return are denoted by "paragraph" breaks.

Attach Call (Or Outer Call From A "New" Process) (Figure 3)

The USER issues a (1) call attach ("henry","tape",...).

The IOSW (2) calls the ATM to retrieve the pointer to the EPV associated with the AT entry for "henry".

The ATM, unable to find an AT entry for "henry", returns a pointer to the EPV for the NF and returns to the IOSW.

The IOSW uses the pointer returned by the ATM to retrieve the link to the outer module (in this case, the NF) and uses the link to forward the call, in effect, issuing a (3) call nf$attach("henry","tape",...).

The NF (4) calls the TTM to retrieve the pointer to the EPV for the tape DIM.

The TTM, noting that the pointer to the EPV for the tape DIM is null (5) calls the EPVMK.

The EPVMK constructs the EPV for the tape DIM and returns a pointer to it to the TTM.

The TTM tables the pointer and returns a copy of it to the NF.

The NF (6) calls the ATM to create an AT entry for "henry" containing a pointer to the EPV for the tape DIM.

The ATM makes the entry in the AT as requested and returns to the NF.

The NF re-issues (7) the user's attach call, to the IOSW.

The IOSW (8) calls the ATM to retrieve the pointer to the EPV in the AT entry for "henry".

The ATM, this time, finds and entry for "henry" and returns the associated pointer to the EPV for the tape DIM.

The IOSW uses the pointer returned by the ATM to retrieve the link to the outer module (in this case, the tape DIM) and uses the link to forward the call, in effect, issuing a (9) call tape$attach("henry","tape",...).

Outer Calls Other Than An Attach (Figure 4)

The USER issues (1) a call attach ("henry", "tape",...).

The IOSW (2) calls the ATM to retrieve the pointer to the EPV associated with the AT entry for "henry".

The ATM finds the entry for "henry" and returns the associated pointer to the EPV for the tape DIM.

The IOSW uses the pointer returned by the ATM to retrieve the link to the outer module (in this case, the tape DIM) and uses the link to forward the call, in effect, issuing (3) a call tape$attach("henry","tape",...).

Design Considerations

It is generally recognized that it is desirable to make it easy for the user to direct a file to different device types for different runs. For example, this need arises in the development of large system programs in which fluctuations in design may entail alterations to the source and destination of given files. Particularly, in debugging, it is convenient to direct intermediate files to a printing device, rather than to their ultimate destinations. Fluctuations in the availability of a given device may require that frames be directed to alternative devices if the program is to be run at a given time.

In systems like IBSYS and GECOS, this has been accomplished by specifying a file code in a preassembled file control block; then at run-time, a control card specified a device in association with the file code. Although this technique achieved the desirable goal of allowing the user, without re-assembly to re-direct his file to any device consistent with the parameters of the file control block; it inflicted a clumsy interface on the user, namely in creating the file control block.

It should be noted that the IBSYS/GECOS technique depends strongly on the batch concepts of loading, control-cards, run-time, etc. In MULTICS, time-shared operation is, at least, as important as batch; and, in on-line operation, there is no concept of loading and interpreting control cards. Hence, the IBSYS/GECOS technique, is not, as such, suitable for MULTICS. One approach could have been to adopt the IBSYS/GECOS technique for batch jobs and invent another for on-line jobs. This approach is not desirable for two reasons: 1) Generally, the aim is for comprehensive solutions covering all the cases; 2) It implies an undesirable distinction between batch and on-line jobs.

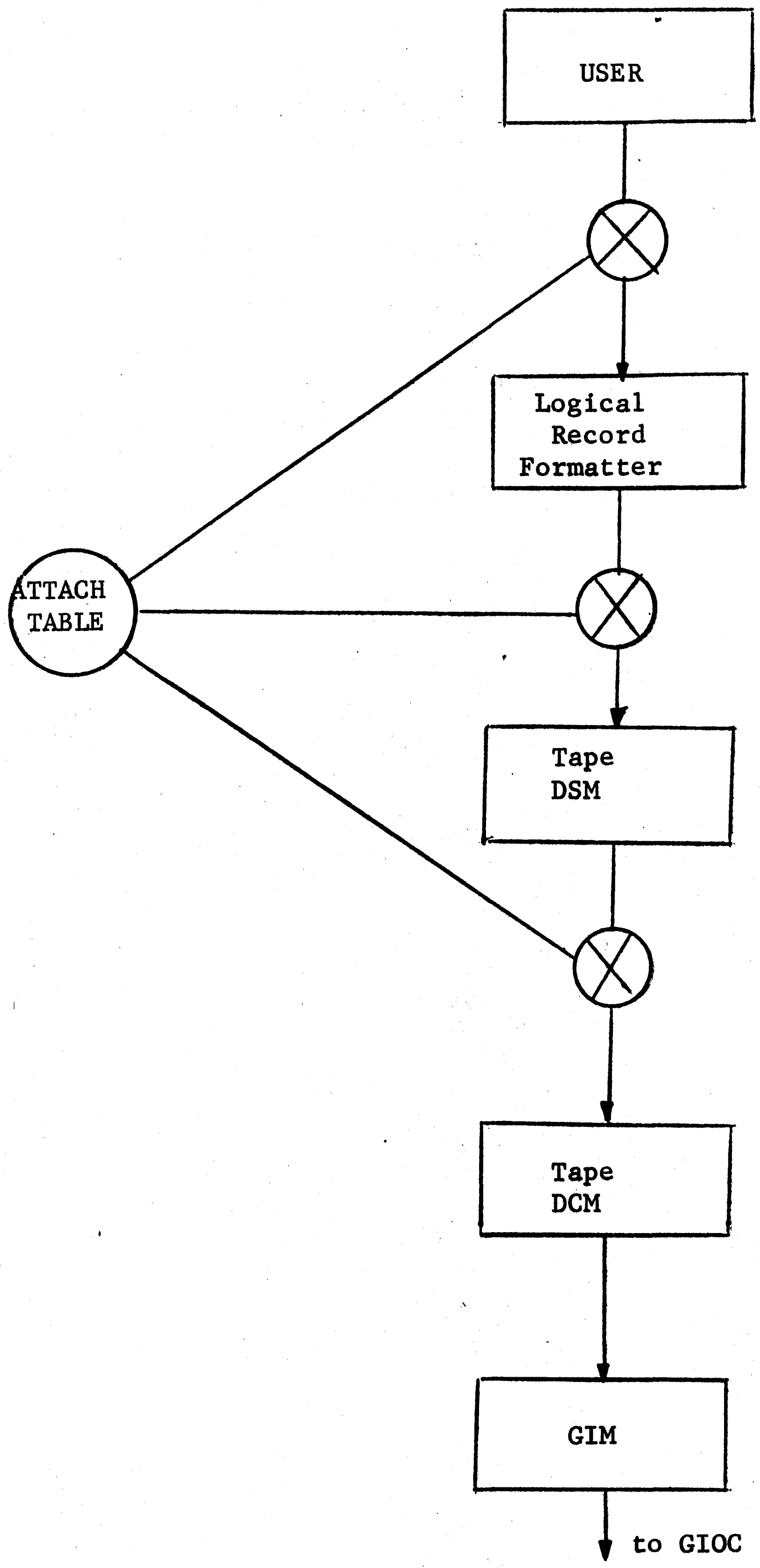
The design adopted for the MULTICS I/O system is equally suitable for batch and on-line jobs. The solution, of course, was to place most of the program's device dependence in the attach call, which as well as other functions, assumes the role of the file control card of the IBSYS/GECOS technique. The adopted solution explains the need for the I/O Switch interface between the user and the rest of the I/O system. Since the attach call specified the device as well as other device oriented specifications, in association with an ioname, some module needs to forward the other outer calls, which do not specify device oriented parameters but instead the ioname, to the proper module.

When this design was first incorporated in the system, it was natural to think in terms of one interface that handled attach calls and set up the correspondence between the ioname and the module which should receive control on later calls; and another, the I/O Switch, which would route the other calls to the proper module. Under that scheme, the attach logic was the one piece of undistributed logic. (Under the distributed logic modularization adopted by the I/O system, one module representing a device contained all the knowledge of that device required to operate it.)

Later, it was realized that if a module (the Not Founder) were added to which the IOSW routed control whenever an ioname was unknown to it, then that module could update the Attach Table with, in particular, the ioname-device interface module association. Having done so, it would re-issue the attach call to the IOSW which would now be able to route the attach call to a proper module of the I/O system. Thus, the Not Founder 1) makes it possible to eliminate the special attach interface to the user, and 2) at the same time, makes it possible to distribute the attach logic.

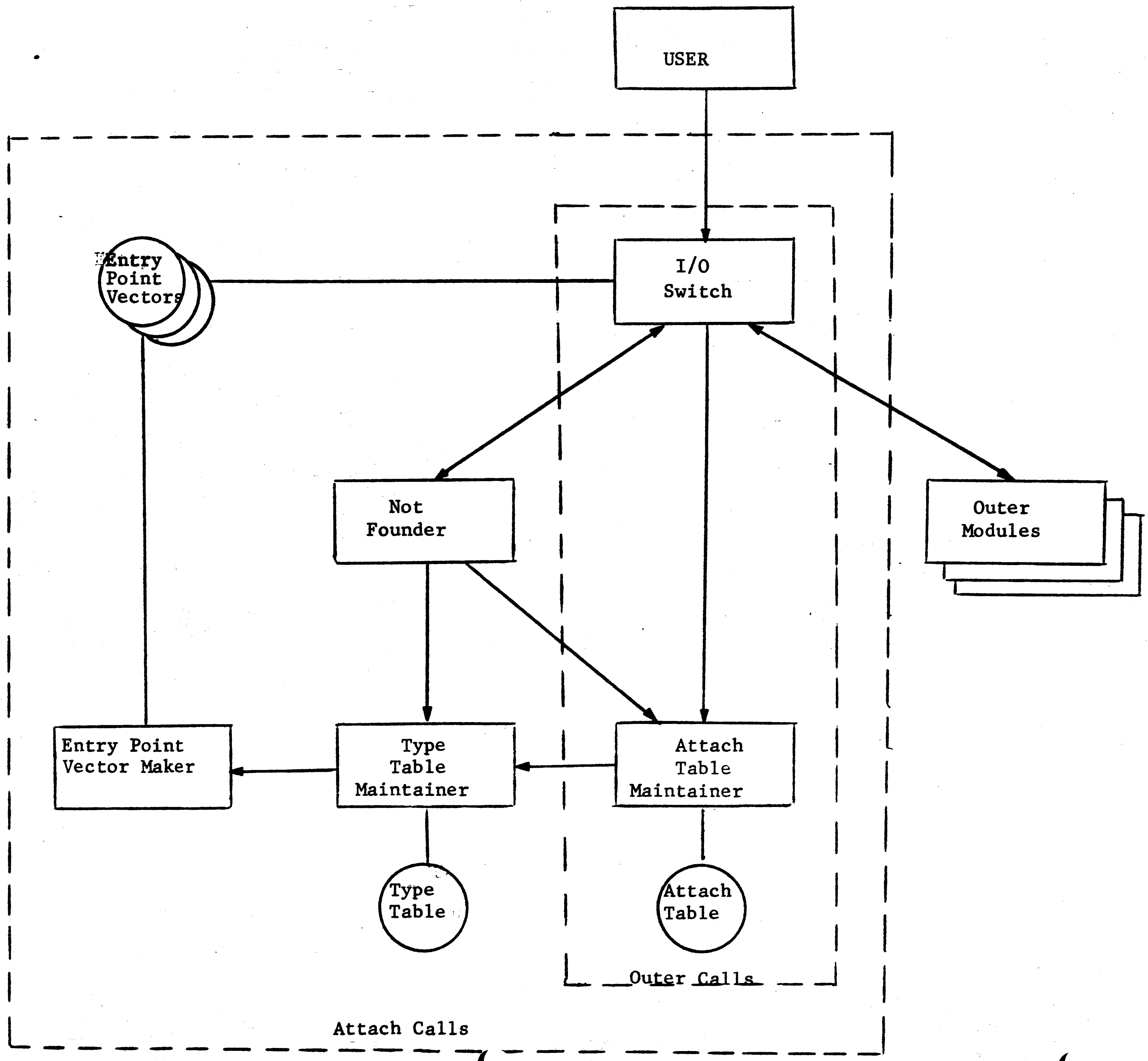
It is desirable to provide the user with the means to conveniently add and replace modules of the I/O system. The I/O system provides this capability, therefore it does not know, in advance, the segment names of all the modules to which it may be required to route control. Hence, a module (the Entry Point Vector Maker) is required to make it possible to dynamically link to unknown segments. Since it has to be there for that purpose, it might just as well be used to link dynamically to known modules. This saves considerable code in the I/O Switch and results in less probability of the I/O Switch being paged out from mere bulk. For example, 30 calls per module and 15 modules represent 450 calls. With an average of 5 arguments per call, this expands to approximately 9000 lines of code plus linkage. On the other hand, the EPVs for the same number of calls and modules require only 900 lines.

The desire to make it easy for the user to replace modules of the I/O system accounts for propagating the user interface into the I/O system as deeply as it will reasonably go. Since, many of the internal modules of the I/O system need to call some other module of the I/O system depending on the software path determined by the attach call, it follows that the IOSW is a convenient interface for them as well as the user.



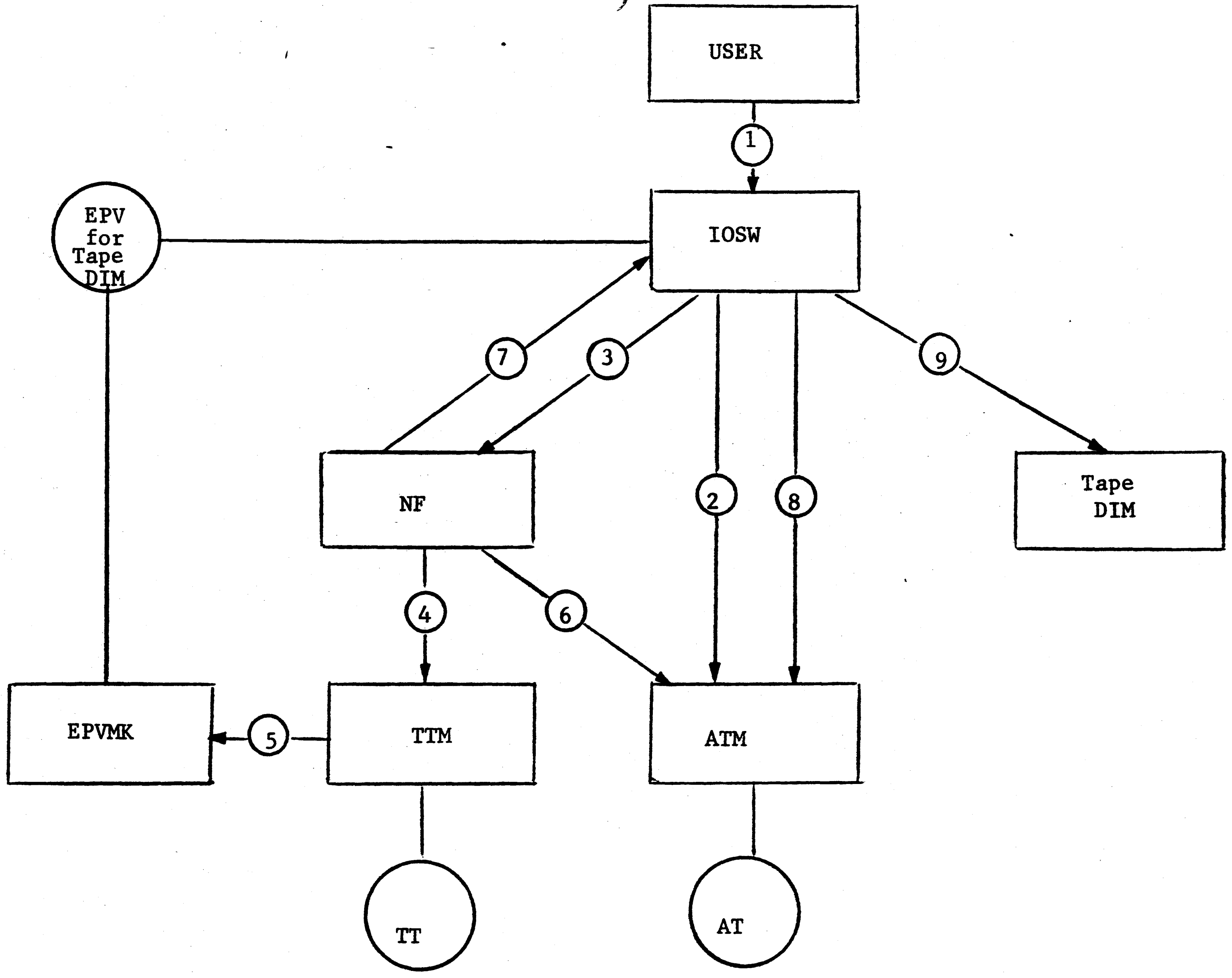
Typical Software Path

Figure 1



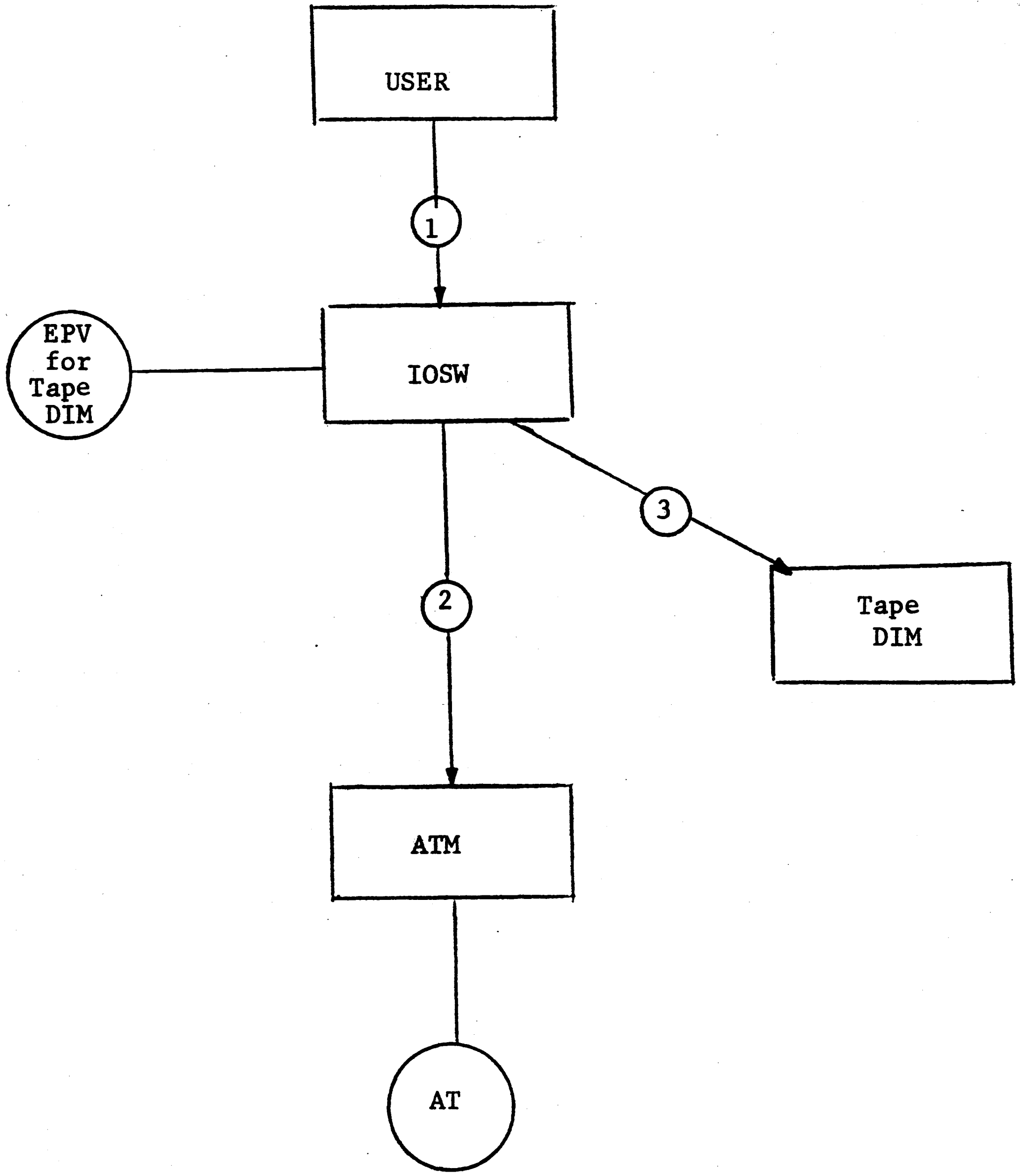
Block Diagram Of Switching Complex

Figure 2



Attach Call Or First Outer Call In A "New" Process

Figure 3



Outer Call Other Than an Attach
(Other Than First Outer Call From "New" Process.)

<u>Figure 4</u>