To: Distribution

From: M.D. MacLaren

Date: May 15, 1974

Subject: New I/O System Features

This MTB describes several new features proposed for The Multics I/O System. A design review will be held in due course. If approved they will probably appear in Release 2.1 of Multics. Send comments to me at CISL or by Multics mail (MacLaren Multics).

Description of the new features is in the form of a draft MPM documentation. A few additional comments may be helpful.

1. New iox_ Entries

These entries permit splicing an attachment into an existing chain (e.g. between user_i/o and the user's console) and facilitate the writing of modules whose attachments have some of the characteristics of a synonym attachment.

The facilities for cross-ring synonyms (already described and reviewed) may also be part of Release 2.

2. Record-Stream I/O Module and PL/I I/O Routines

The aim of these facilities is to allow a user, especially a PL/I user, to use only record I/O. The record-stream module also attacks the general problem of what to do with a program that expects record files when the data is in stream files or vice-versa. A gap exists in the modules' specification. Nothing is said about the treatment of pages. Suggestions on this point are especially welcome.

The PL/I I/O routines cope with the specific problem of handling varying strings in PL/I record I/O. For technical reasons, a PL/I record I/O statement must treat a varying string variable as a block of storage determined by its maximum length.

Multics Project internal working documentation. Not to be reproduced or distributed outside the Multics Project.

Thus

```
dcl s char (10000)
s = "a"
write file (f) from (s);
```

writes out a 10,000 byte record, and

```
read file (console_input) into (s);
```

raises the record condition because the line obtained from the console is shorter than 10,000 bytes. Also, the first four characters of the line are placed in the length field of s.

The proposed routines use the value of a varying string as the record. It is worth noting that the output routines are not strictly necessary. One could use a routine like:

```
write: proc (f,s2);
   declare f file, s2 char (*) /* not varying */;
   write file (f) from (s2);
   end;
```

The call

```
call write (f, (s));
```

writes out the value of the string s, whether s is varying or nonvarying, indeed even if it is an expression. Providing a special output routine for varying strings avoids the dummy argument, avoids having to explain to the user how he can write his own routine, and gives the appearance of a balanced facility. (The input routines cannot be done within PL/I).

3. Improvements to vfile

The improvements are:

1. recovery from interrupted openings (crashes) for indexed files as an automatic feature.

2. a file_utility command that prints information about files and adjusts files after interrupted openings

3. a limited form of inter-process sharing for indexed files. The sharing has been limited to that which can be supported by a single lock in the the file and a very simple locking strategy.

4. the control operation accepts an order "seek-head" that finds the first record whose key has a specified relation to a given key. This is required by COBOL.

SUBSYSTEM WRITERS GUIDE REFERENCE DATA FOR NEW iox_ ENTRIES--GOES IN SECTION 3.8, WRITING AN I/O MODULE

Generalized Synonyms

The I/O system supports a more general form of synonym than that described in the MPM Section, The Multics I/O System. This general form of synonym is useful in writing an I/O module that is to monitor some I/O activity or that modifies slightly the behavior of another module.

To establish a switch s1 as a generalized synonym for another switch s2, call the entry iox_$set_syn. This sets all entry values in s1's control block to a call forwarding routine that will forward calls to the corresponding entry values in s2, copying the argument list with the control-block pointer replaced by a pointer to s2's control block. On return from iox_$set_syn, set those entries that should not be call forwarders. Of course, when all changes have been made to s1's control block, you must call iox_$propagate.

The call to iox_$set_syn also established a notification entry value, p, for the relationship between s1 and s2. As long as s1 remains a generalized synonym for s2, p will be called whenever changes are made to s2's control block. (This call is arranged by iox_$propagate). The entry p must be external and must match the declaration

```
declare p entry (ptr);
```

When called, it is given a pointer to s1's control block. This enables p (if properly programmed) to determine what switches are involved and what action (if any) to take when s2 is changed.

To disestablish a generalized snyonym, call iox_$reset_syn. This changes call forwarding entry values to iox_$err_no_operation. The notification procedure will no longer be called.

Moving Attachments

It is possible to move an attachment from one I/O switch to another. This affects the switches but not control blocks maintained by the I/O module. For example, moving the console attachment from user_i/o to some other switch, will have no effect on the terminal.

To move an attachment, call iox_$move_attach. Note that the rules for writing an I/O module prohibit an attach routine from moving the attachment of a switch s in order to attach s.

PROPOSED SPECS FOR NEW iox_ENTRIES

Name: iox_$move_attach

This entry moves an attachment from one I/O switch, s1, to another I/O switch, s2. The switch s2 must be in the detached state when the entry is called. If it is not, the code error_table_$not_detached is retured and no change is made to s1 or s2.

Moving the attachment moves the attach description and open description of s1 to s2. All pointer values and entry values are copied from the control block of s2 to the control block of s2. (These values are enumerated in the Subsystem Writer's Guide Section, The I/O Control Block). Attach and open data blocks maintained by the I/O Module (if s1 is attached) are not affected. Finally, s1 is set to the detached state and iox_$propagate is called for both s1 and s2.

Usage:

```
declare iox_$move_attach entry (ptr,ptr,fixed(35));
call iox_$move_attach (iocb_ptr_1,iocb_ptr_2,code);
```

1. iocb_ptr_1 points to the control block for s1 (Input)
2. iocb_ptr_2 points to the control block for s2 (Input)
3. code is a system status code (Output)

Name: iox_$set_syn

This entry establishes an I/O switch, s1, as a generalized synonym for another I/O switch, s2. It is useful in writing certain types of I/O modules. See the discussion of generalized synonyms in the Subsystem Writer's Guide Section, Writing an I/O Module.

The switch s2 must not be a synonym or generalized synonym for s1, either directly or via a chain. If it is, the code error_table_$circular_syn is returned, and no change is made to s1.

The entry sets all entry values in s1's control block to a routine that will forward calls to the corresponding entry variable in s2's control block. No other changes are made to s1 and no changes are made to s2.

A call to iox_$set_syn also establishes an entry that will be called by the I/O system whenever changes are made to s2's control block. This entry must be external, and it must match the declaration

```
declare notify_proc entry (ptr);
```

When a change is made to s2, the I/O system (via iox_$propagate) makes the call

```
call notify_proc(iocb_ptr_1);
```

where iocb_ptr_1 points to s1's control block. If the notify_proc procedure makes any changes to s1's control block, it is responsible for calling iox_$propagate for s1.

Usage:

```
declare iox_$set_syn (ptr,ptr,entry,fixed (35));
call iox_$set_syn (iocb_ptr_1,iocb_ptr_2,notify_proc,code);
```

1. iocb_ptr_1 points to the control block for s1. (Input)
2. iocb_ptr_2 points to the control block for s2. (Input)
3. notify_proc is the procedure to be notified whenever changes are made to s2's control block. (Input)
4. code is a system status code. (Output)

Name: iox_$reset_syn

This entry disestablishes the relationship of an I/O switch, s1, as a generalized synonym for another I/O switch. All entry variables in s1's control block that are set to the call forwarding routine are reset to iox_$err_no_operation.

If the switch s1 is not, in fact, a generalized synonym for another switch, the code error_table_$not_attached is returned, and s1 is not changed.

Usage:

```
declare iox_$reset_syn entry (ptr,fixed(35));
call iox_$reset_syn (iocb_ptr_1,code);
```

1. iocb_ptr_1 points to the s1's control block. (Input)
2. code is a system status code. (Output)

PROPOSED SPECS FOR record_stream I/O MODULE

I/O Module

Name: record_stream

This I/O module attaches an I/O switch to a target I/O switch so that record I/O operations on the attached switch are translated into stream I/O operations on the target switch, or so that stream I/O operations on the attached switch are translated into record I/O operations on the target switch. In this way a program that uses only record I/O may process unstructured files and do I/O from/to the console. Similarly a program that used only stream I/O may process some structured files.

Entries in this module are not called directly by users: rather the module is accessed through the I/O system. See the MPM Section, The Multics I/O System, for a general description of the I/O system.

Attach Description

The attach description has the following form:

```
record_stream switchname2 -options-
```

1. switchname2 is the name of the target I/O switch. It need not be attached when this attachment is made.
2. options are chosen from the following and control the transformation of records into a stream of bytes or vice-versa.

   -line means that a line is to be transformed into a record (or sequence of records) or that a record is to be transformed into a line.

   -no_line means that there is no relation between lines and records. This option must not occur if -line does occur.

   -trim means that in transforming a record trailing blanks are deleted.

   -length n means that the stream of bytes is converted to a sequence of records each of which has length n.

If neither -line or -no_line occurs, -line is supplied by default.

Opening

The attached I/O switch may be opened for stream_input, stream_output, sequential_input, or sequential_output. The implications of the opening mode are as follows (in addition to the usual implications described in the MPM Section, The I/O System).

1. Stream_input. The target I/O switch must be attached and closed. It is opened for sequential_input. The sequence of records read from the target switch is transformed into a stream of bytes which are transmitted to the calling program by get_line and get_chars operations. The operations read_length and read_record are used to read the records from the target switch.

2. Stream_output. The target I/O switch must be attached and closed. It is opened for sequential_output. The stream of bytes written to the attached switch by the put_chars operation is transformed into a sequence of records which are written to the target switch by use of the write_record operation. At least one of the -line and -length options must occur in the attach description.

3. Sequential_input. The target I/O switch must be open for stream_input, open for stream_input_output, or attached and closed. In the last case, it is opened for stream_input. The stream of bytes read from the target switch is transformed into a sequence of records, which are transmitted to the calling program by read_record operations. If the attach description specifies -line (explicitly or by default), the get_line operation is used to read bytes from the target switch. If the attach description specifies -no_line, the get_chars operation is used to read bytes from the target switch. In this case the -length option must occur.

4. Sequential_output. The target I/O switch must be open for stream_output, open for stream_input_output, or attached and closed. In the last case, it is opened for stream_output. The sequence of bytes written to the attached switch by the write_record operation is transformed into a stream of bytes which are written to the attached switch by use of the put_chars operation.

Transformations

The transformation from record to stream from can be described in terms of taking records from a record switch and giving bytes to a stream switch, and similarly for stream to record. Which switch is the record switch and which the stream switch depends on the opening mode as explained under "Opening". The transformation is controlled by the options in the attach description. The details are as follows. (Note that a record is a string of bytes.)

1. Record to stream, -line

   A record is taken from the record switch, a new line character is appended,and the resulting string is given to the stream switch.

2. Record to stream, -line -trim.

   A record is taken from the record switch, trailing blanks (if any) are deleted from it, a new line character is appended, and the resulting string is given to the stream switch.

3. Record to stream, -no_line

   A record is taken from the record switch and given to the stream switch without modification.

4. Record to stream, -no_line -trim

   A record is taken from the record switch, trailing blanks (if any) are deleted from it, and the resulting string is given to the stream switch.

5. Stream to record, -line

   A line (string of bytes ending with a new line character) is taken from the stream switch, the new line character is deleted, and the resulting string is given to the record switch.

6. Stream to record, -line -length n

   A line is taken from the stream switch. The new line character is deleted. Blanks are appended so that the length, m, of the resulting string is a multiple of n, but no more than necessary (possibly none) are appended. The resulting string is divided into (m/n) records, and the records are given to the record switch.

7. Stream to record, -no_line -length n

   To form a record, n bytes are taken from the stream switch and given to the record switch as one record.

Buffering

The I/O module may hold data in buffers between operations when the switch is opened for stream_output or stream_input, or for sequential_input in the "-line -length" case.

Close Operation

The I/O module closes the target switch if an only if it opened it.

Position Operation

Only positioning to the beginning of file or end of file and skipping forward are supported, and they are only supported to the extent the attachment of the target I/O switch supports them.

Control and Modes Operations

These are supported for sequential_input and sequential_output in the sense that they are passed along to the I/O module for the target switch. Note that the target switch will be open for stream_input, stream_output, or stream_input_output.

Error Codes

In addition to the error codes specified in the description of iox_ for the various I/O operations, this I/O module returns error codes returned by the I/O module for the target switch.

Example

Given a file, foo, containing lines of varying lengths but less than or equal to 136 (including the new line character), the following commands open a switch, s3, for stream_input so that all the lines are padded with blanks to be length 136.

```
io_call attach s1 vfile_ foo
io_call attach s2 record_stream s1 -line -length 135
io_call attach s3 record_stream s2 -line
io_call open s3 stream_input
```

As a consequence of these commands, the I/O switch s2 is opened for sequential_input, and the I/O switch s1 is opened for stream_input.

PROPOSED SPECS FOR NEW PL/I-I/O ROUTINES

Subroutine

Name: read_string_

This routine is for use with PL/I files. It reads the next record in a PL/I file and assigns the record (as a string) to a varying string variable. In this way, a user may conveniently read records of unknown length.

Usage

```
declare read_string_key_ entry(file,char(*)varying);
call read_string_ (f,s);
```

1. f is the file. It must be open for sequential input, keyed sequential input, sequential update, or keyed sequential update. (Input)

2. s is the string to which the record is to be assigned. (Output)

Subroutine

Name: read_string_key_

This routine is for use with PL/I files. It reads a record with a specified key, and assigns the record (as a string to a varying string variable. In this way a user may conveniently read records of unknown length.

Usage

```
declare read_string_key_ entry (file,char(*)varying,
     char(256)varying);
call read_string_key_ (f,s,key);
```

1. f is the file. It must be open for keyed sequential input, keyed sequential update, direct input, or direct update. (Input)
2. s is the string to which the record is to be assigned. (Output)
3. key is the key. (Input)

Subroutine

Name: write_string_

This routine is for use with PL/I files. It writes the current contents of a varying string variable as a record (not the whole block of storage occupied by the variable).

Usage

```
declare write_string_ entry(file,char(*)varying);
call write_string_(f,s,);
```

1. f is the file. It must be open for sequential output. (Input)
2. s is the string to be written. (Input)

Subroutine

Name: write_string_key_

This routine is for use with PL/I files. It wrties the current contents of a varying string variable as a record (not the whole block of storage occuped by the variable). The record is written with a specified key.

Usage

```
declare write_string_key_ entry (file,char(*) varying,
     char(256)varying):
call write_string_key_ (f,s,key):
```

1. f is the file. It must be open for keyed sequential output, keyed sequential update, direct output, or direct update. (Input)
2. s is the string to be written. (Input)
3. key is the key. (Input)

PROPOSED SPECS FOR vfile_

I/O Module

Name: vfile_

This I/O module supports the I/O from/to files in the storage system. All logical file types are supported.

Entries in this module are not called directly by users; rather, the module is accessed through the I/O system. See the MPM section, the Multics I/O system, for a general description of the I/O system, and see the MPM section, file I/O, for a discussion of files.

Attach Description

The attach description has the following form:

vfile_ pathname -option_1-...-option_n-

1. pathname is the absolute or relative pathname of the file.

2. option_i can be chosen from the following list of options.

-extend specifies extension of the file if it already exists. This option is only valid with openings for output or input_output.

-set_bc specifies that each out_chars operation will set the bit count of the appropriate segment of the unstructured file to which the I/O switch is attached. This option is only valid with openings for stream_output or stream_input_output.

-share -wait_time- allows an indexed file to be open in more than one process at the same time, even though not all openings are for input. (See Multiple Openings below). This option is only valid with openings for direct_input, direct_update, or direct_output with -extend. The wait_time, if specified, is the maximum time in seconds that this process will wait to perform an operation on the file. A value of minus one means the process may wait indefinitely. If no wait_time is given, a default value is used.

To form the attach description actually used in the attachment, the pathname is expanded to obtain an absolute pathname.

Opening and Access Requirements

All opening modes are supported. For an existing file, the mode must be compatible with the file type. (See the MPM Section, File I/O). The mode must be compatible with any options in the attach description.

If the opening is for input only and without the -share option, only "r" access is required on the file. In all other cases "rw" access is required on the file.

Rewrite Operation

If the file is a sequential file, the new record must be the same length as the replaced record. If not, the code returned is error_table_$long_record or error_table_$short_record.

Delete Operation

If the file is a sequential file, the record is logically deleted, but the space it occupies is not recovered.

Modes Operation

This operation is not supported.

Control Operation

The order "seek_head" is accepted when the I/O switch is open for keyed_sequential_input or keyed_sequential_update. For this order the info_ptr argument must point to a structure of the following form:

```
dcl  1  info based (info_ptr),
       2  relation_type fixed,
       2  n fixed,
       2  search_key char (* refer (n));
```

The operation locates the first record with a key whose head has the specified relation with the given search_key. The next record position and (for keyed_sequential_update) the current record position are set to the record. If no such record exists, the code, error_table_$no_record is returned.

The head of a record's key is the first n characters of the key, the key being extended by blanks if it has fewer than n characters. The allowed values for info.relation_type are:

0	head = search_key
1	head >= search_key
2	head > search_key

Multiple Openings

It is possible to have or attempt to have multiple openings of the same file, that is to have two or more open I/O switches attached to the same file. These switches might be in the same process or in different processes. With respect to the effects of multiple openings, the various opening modes can be divided into four classes (explained below). Multiple openings in which the opening modes are in more than one class are invalid, as are multiple openings within certain classes. The vfile_ module prevents some cases of multiple opening, the code error_table_$file_busy being returned by the open operation. In cases where an invalid multiple opening does occur, I/O operations will cause unpredictable errors in the processes involved, and the contents of the files may be damaged.

The classes of multiple openings are:

1. Openings for input without the -share option.

Any number of openings in this class are allowed. The existence of an opening in this class never causes damage to the file. When this class of opening is attempted, the existence of all class 2 and 3 openings and some class 4 openings will be detected for structured files.

2. Openings for output or input_output without file extension.

Only one opening is allowed. The existence of another opening is never detected when this class of opening is attempted. The file is simply replaced by an empty file of the appropriate type. If the file was already open with an opening of any class except (1), the contents of the new file will probably be damaged.

3. Openings for update without the -share option and for output or input_output without the -share option and with file extension.

Only one opening of this class is allowed. For structured files, multiple openings within the class are detected. An invalid multiple opening involving an opening of this class and other openings of class 4, may be detected. If not, the only effect is that the class 3 opening locks the file for the entire opening.

4. Openings with the -share option.

(This applies to direct_input, direct_update, and direct_output with -extend only). Any number of openings of this type are allowed. When a process performs an operation on the file, the file is locked. Other processes attempting an operation while the file is locked will wait up to the limit specified by the wait_time option in the attach_description. If the operation is not carried out because of the wait_time limit, the code error_table_$file_busy is returned.

There are two system status codes that pertain only to class 4 openings: error_table_$asynch_deletion and error_table_$asynch_insertion. The first is returned by the read_record, read_length, and rewrite_record operations when a record located by a seek_key operation has been deleted (by an operation in some other opening). The second is returned by write_record when a record with the key for insertion (defined by a seek_key operation) has already been inserted (by some other opening).

Interrupted Openings

If a process opens a file and terminates without closing the file, the file may be left in an intermidiate state that prohibits normal I/O operations on the file. The exception is openings for input only. The details depend on the particular type of file as follows.

1. Unstructured files.

In general, the bit count of the file's last segment will not be properly set. This condition is not detected at subsequent openings, and part of the file's contents may be overwritten or ignored. Use the command file_utility to properly set the bit count.

2. Sequential file.

In general, certain descriptors in the file and the bit count of the files' last segment will not be properly set. This condition is detected at a subsequent open, and the code error_table_$file_busy is returned. Use the command file_utility to properly adjust the file.

3. Indexed files.

In general, the bit counts of the files' segments will not be properly set,and the files' contents will be in a complex intermediate state (e.g. a record will be deleted but not its key in the index). This situation is detected at a subsequent open; and, unless the opening is for input without the -share option, the file is automatically adjusted. If this situation is detected by an opening for input without the -share option, the code error_table_$file_busy is returned. The command file_utility may be used to properly adjust the file.

When an indexed file is adjusted, the interrupted operation (write_record, rewrite_record, or delete_record), if any, is completed. However for write_record or rewrite_record, it may happen that the bytes of the record are potentially incorrect. (Everything else will be correct). In this case an error message is printed on the console. The user can rewrite or delete the record as required.

Inconsistent Files

The code error_table_$bad_file (console message: "File is not a structured file or is inconsistent") may be returned by operation on strucured files. It means that an inconsistency has been detected in the file. Possible causes are:

1. The file is not a structured file of the required type.

2. A program accidentally modified some words in the file.

In the last case an earlier consistent copy of the file should be reloaded.

PROPOSED SPECS FOR file_utility COMMAND

Name: file_utility.fu

This command provides a few services related to storage system files as supported by the I/O module vfile_.

Usage:

file_utility pathname action -option_1-...-option_n-

1. pathname is the pathname (absolute or relative) of a file in the storage system.

2. action is -status or -adjust. Its meaning is explained below.

3. option_i depends on action as explained below.

Action: -status

This action prints the file's apparent type (unstructured, sequential, or indexed) on the console. For structured files, information on current openings of the file or an interrupted opening is printed. For indexed files, the following statistics are also printed:

1. the number of records in the file, including zero length records.
2. the total length of the records.
3. the number of blocks in the free-space list for records.
4. the height of the index tree. (For an empty file the height is zero).
5. the number of nodes in the index tree.
6. the total length of all keys.

Additional information about a file may be obtained through the status command.

No options are allowed with the -status action.

Action: -adjust

An interrupted opening of a file may leave it in an intermediate state that prohibits normal I/O operations on the file. (See the write-up of the vfile_ I/O module for details.) This action adjusts the file to an appropriate state. It may be used on unstructured files whether or not their current state resulted from an interrupted opening. The allowed options are:

-set_bc — The file must be unstructured. The bit count of the file's last segment is set to the last nonzero byte in the segment. The last segment is taken to be the last segment with nonzero contents. Any segments beyond it are deleted.

-use_bc -n- — The file must be unstructured. It is truncated to the byte specified by the bit count of segment n. (Segments are numbered 0,1,2...). If n is not given, it is taken to be the files last nonzero segment.

-nl — The file must be unstructured. It is truncated to the last new-line character in the file, and the bit count of the last segment is set accordingly.

If no option occurs, the file must be structured. A sequential file is adjusted by truncating it to the last complete record in the file. An indexed file is adjusted by completing an interrupted operation. See the write-up of the I/O module vfile_ for details.