TO: Distribution

FROM: M. D. MacLaren

DATE: December 7, 1973

SUBJECT: More on the New I/O System

There have been further design reviews and consequent design changes. As a result, the existing documentation (MSB-113, MTB-010), which was difficult enough to use before the excitement, is now obsolete.

This technical bulletin attempts to give a brief discussion of each important feature and each controversial issue. Unfortunately, complete documentation (in the form of draft MPM documentation) will not be available until January. The author apologizes for this situation.

## GOALS OF THE NEW SYSTEM DESIGN

The new system design is intended to provide efficient and clean support for device independent I/O - both record and stream - without doing any damage - except philosphical - to device dependent I/O. It corrects two major problems in the system: excessive cost in going through the switch and lack of device independent operations. It adds support for record I/O which will be used by PL/I, Fortran, and COBOL.

## COMPATIBILITY

Although ios_ will be phased out from Honeywell software, it and old DIMs can exist indefinitely as installation maintained or private subroutines without conflict with the new system. Note, however, that some old DIMs (e.g. tape_) depend on internal interfaces that are unlikely to be maintained indefinitely.

To avoid confusion between new and old, the new switch entries are of the form "iox_$..." and the new DIMs are called "iomodules". The term "ioname" is used instead of stream name.

## THE NEW SWITCH

An I/O operation other than attachment is performed by a call of the form:

```
call iox_$op (iocb_ptr, arg2,..., code);
```

(design change suggested by D. Clark). Here op is the name of the particular operation, code is a standard Multics status code, and iocb_ptr is a pointer previously obtained by a call of the form

```
call iox_$find_iocb (ioname, iocb_ptr, code);
```

Thus the user need not know the form of an I/O control block. (System routines, such as the operators for PL/I I/O, may call directly using the iocb).

The entry iox_$op is a trivial call forwarder that, in effect, turns the call into

```
call iocb_ptr -> iocb.op (iocb_ptr, arg2, ..., code);
```

The new switch has the following advantages over the ios_ switch.

1) The table look_up on the ioname is avoided.

2) The argument list does not have to be transformed before forwarding it to the I/O module.

3) The variable transfer vector in the iocb lets an I/O module handle the separation of attach and open and the various modes of opening without testing at each call. It also permits an I/O module to be coded entirely in PL/I.

VERSION_NUMBER (design change)

The iocb will contain a version number.

THE_STANDARD_I/O_NAMES (design change)

To permit access to user_input, user_output, and error_output without a call to iox_$find_iocb, three external pointer variables will be provided:

```
iox_$user_input
iox_$user_output
iox_$error_output
```

They will be initialized during process initialization. Thus, where with ios_ one would use

```
call ios_$write_ptr (workspace, offset, nelem),
```

with iox_ one will use

```
call iox_$put_chars (iox_$user_output,
     buff_ptr, buff_len, code);
```

The second call is inherently more efficient than the first, because its argument list does not have to be transformed before passing it to the IIM (iomodule).

ATTACHING I/O NAMES (design change)

To attach an ioname in the new I/O system, use the command

```
attach ioname attach_description
```

or one of the calls

```
call iox_$attach_iocb (iocb_ptr, attach_description, code);

call iox_$attach_ioname (ioname, iocb_ptr,
     attach_description, code);
```

In the last case, iocb_ptr is an output parameter.

The attach_description may also be used as the title in a PL/I open statement. In all cases its form is

```
module_name  option-1...option-n
```

where the options depend on the particular module. If module_name is an entry name (i.e. contains no "<" or ">"), the module is found by the search rules. Otherwise it is found by using expand_path_.

A system module has a name ending in "_". In all cases the attachment is actually made by an entry whose name is "module$module attach", (e.g. "tape_$tape_attach"). The general form of the call is

```
call module$module attach (iocb_ptr, option_array,
     command_switch, code);
```

The option array is char(*)(*) varying and contains option-1...option-n from the attach description. The command_switch is "1"b if the module was called from the attach command and is "0"b otherwise.

The aim of all this is to support attachment from command level, by a PL/I title option, or by subroutine call, to use a single entry, and to use the same character-string (no bit(36) or fixed bin(2) arguments) to describe attachment in all cases. The attach_description (perhaps expanded) is available later through the iocb and is used by print_attach_table.

Full implementation of this uniformity requires utility routines for 1) parsing an attach_description according to command processor conventions (blanks and quotes) and 2)

synthesizing an attach description from its parsed options (e.g. quoting options containing blanks). This degree of implementation will not be available in the first release.

## THE NEW IOMODULES

The following iomodules will be provided in the first release.

svn_ synonym attachments.

tty_ the tty DIM as currently implemented.

discard_ sink for output.

ntape_ supports the opening modes sequential_input, sequential_input_output and sequential_output for tapes with each physical record interpreted as a logical record.

vfile_ supports all opening modes for files in the storage system. The treatment of files is as described in MSB-113 and MTB-010.

Note that svn_ and tty_ can also be attached by calling ios_$attach for svn and tw_. This is part of the compatibility feature.

## OPEN-CLOSE-DETACH

The operation detach is renamed detach_iocb (i.e., call iox_$detach_iocb (...)). Otherwise these operations are as described in MSB-113. Note that ios_$attach corresponds to iox_$attach_iocb followed by iox_$open, and ios_$detach corresponds to iox_$close followed by iox_$detach_iocb. An iomodule may automatically perform opening as part of attaching and detaching as part of closing, but it is generally advantageous to separate the functions.

## THE NEW OPERATIONS

The following operations are defined for the initial release of the system:

| | |
|---|---|
| detach_iocb | rewrite_record |
| open | delete_record |
| close | position |
| get_line | seek_key |
| read_key | put_chars |
| read_record | control |
| write_record | read_length |
| get_chars | modes |

Except for the modes entry these are as defined in MSB-113, but "_record" has been appended to some names to increase the distinction between stream and record I/O (design change).

The operations read_record and write_record are not supported for stream openings (design change). (See MAB-113, p. 14 for a list of opening modes). A simple interface module can handle the problem of using PL/I or COBOL record I/O with a stream.

The disposition of ios_ entries not included in the new system is as follows.

1) resetread, resetwrite, abort, and order are all handled by control.

2) all other entries pertaining to control of asynchronous I/O are waiting on review of the proper way to do asynchronous I/O in Multics. There is no problem in extending iox_ to handle appropriate asynchronous operations when they are designed.

3) getdelim, setdelim, getsize, setsize, seek, and tell have been deleted because they are not device independent.

DEVICE INDEPENDENCE

The languages and other higher level users of the I/O system cannot afford to have the meaning of the I/O operations be device dependent. In particular they need a standard element size (the 9-bit byte), they need to know that a call to get_line actually returns a line (not say everything up to the first letter), and they need to know that a call to put_chars actually transmits all the data (not just what the DIM is in the mood to accept).

With the old I/O system, this device independence could only be achieved by enforcing standards that go against the spirit of the old system (e.g. don't use setdelim on any DIM that is to be considered standard). Enforcing these standards would require recoding of DIMs and callers. It is cleaner to build the standards into the new system and eliminate entries that violate the standards.

Note that the new system does not prevent device dependent I/O. For example, a device dependent I/O module could support variable element sizes by use of an appropriate mode. However, if in the call

```
get_line(iocb_ptr, buff_ptr, buff_len, rec_len, code);
```

the length of the buffer and the length of the string actually written into the buffer, are given in other than 9-bit bytes, then the I/O module violates the system conventions and can only be used in a restricted way.

In regards to seek and tell, it must be admitted that they can be given a restricted device independent meaning but in this form they are of little value in either stream or record I/O and are expensive to implement.

MODES and CONTROL (design change)

These entries are intended to support I/O to-from terminals and other communications type devices. Setting modes has a persistent effect (e.g. turn off erase and kill). Calling control has a one-time effect (e.g. resetread). Breaks will probably be implemented as a mode; but this will be decided as part of the new tty DIM design - not as part of the iox design. Setting breaks will probably change the behavior of get_chars. It will not affect get_line which always returns an entire line.

The modes entry is equivalent to ios_changemode but the specification is tightened up a bit.

```
declare iox_$modes entry (ptr, char(*),
        char(*), fixed bin(36));

call iox_$modes (iocb_ptr, new_modes,
        old_modes, code);
```

The modes are given as a list of individual modes separated by commas. The list is terminated by the first blank or the end of the whole string. Thus a mode cannot contain a blank. If the argument new_mode contains any illegal mode (for that I/O module) or the argument old_modes is too short to hold the list of old modes, then no modes are set and error_table_$bad_arg is returned. When new_modes is empty, the only effect of the call is to get the old modes.

The control entry is as in MSB-113,

```
call iox_$control(iocb_ptr, order, info_ptr, code);
```

but certain orders are given a standard meaning (and more may be added in the future).

resetread, resetwrite: The meaning is the same as ios_$reset except that info_ptr may be nonnull. In that case it points to a structure

```
declare 1 s,
          2 buff_ptr ptr,
          2 buff_len fixed(21),
          2 rec_len fixed(21);
```

and the canceled input or output will be written into the indicated buffer. In this case error_table_$long_record will be returned if the buffer is too small.

abort: same as ios_$abort.

hardware_status: info_ptr points to bit(72), and a hardware status code is stored.

For resetread, resetwrite, and abort a code of zero is returned if the order is not supported. This will be handled automatically (by propagate) for iomodules that support no orders, i.e. that don't provide a control entry.