TO: MSPM Distribution
FROM: D. R. Widrig
SUBJECT: BF.20.09
DATE: 12/01/67

Slight modifications to some data declarations for the GIM have been made. Specific areas affected are:

1. Hard core 1/O data bases
   a. nadpt
   b. n_phys_chnls
2. Channel Assignment Table (CAT)

Published: 12/01/67
(Supersedes: BF.20.09, 07/10/67)

Identification

GIM - Basic Concepts
D. R. Widrig and S. D. Dunten

Purpose

Sections BF.20.09 through BF.20.13 are a detailed discussion of the internal workings of the entire Hard-core I/O System. Collectively, the modules comprising the Hard-core I/O System are known as the GIOC Interface Module (GIM). This section forms part 1 of the GIM description: see BF.20.02. The GIM is the sole interface between the Multics Device Interface Modules, (DIMs) and the General Input Output Controller (GIOC). As such, it is invested with many system responsibilities such as user security, protection against accidental or willful mis-programming, proper time accounting, awakening I/O-blocked processes, etc. Many of the system responsibilities tend to oppose or conflict with each other. It will be shown that such conflicts figured quite heavily in the design of the GIM and its interfaces to the DIM programmers in the role of GIM users. Knowledge of general GIM philosophy outlined in BF.20.01 is assumed. It will be helpful to refer to BF.20.03, Summary of GIM Calls and Data Bases, while reading this document.

Introduction

Central to the understanding of the operations of the GIM is the notion of a list. We shall define a list as a structured array of symbolic descriptions which direct the GIM to perform a useful service for a user.

For instance, one could imagine a list which tells the GIM to forward-space a tape 3 records and then read the next record into a particular buffer area.

One could program the GIM with the knowledge necessary to perform the above actions. Thus, the user need merely say to the GIM, "perform the forward-space operation 3 times, then do the tape read operation into this buffer". The main problem with this type of scheme is in requiring the GIM to know how to perform all operations for all devices, a massive amount of knowledge. Moreover, as new devices are added, one would be obliged to re-program the GIM to add the new device facilities.

Moving to the opposite pole, one could program the GIM to accept requests in the form of standard GIOC "instructions", the DCW's. In this case, the GIM need know nothing about any device; it merely accepts DCW lists and places them in the appropriate user areas. The major flaw in the above implementation is primarily a question of security.

The question of security arises out of the observation that the GIOC does not operate with the standard appending and protection mechanisms available to the 645 processors. Thus, at the GIOC level, one must deal in I/O transmissions directed to absolute addresses of wired-down or latched core. Also, one has absolutely no hardware means of preventing the GIOC from reading or writing protected areas thus leaving open the possibility for core disasters or equally disastrous security breaches. Clearly, the acceptance of raw DCW's from a DIM call to the GIM is not a viable interface.

The list interface to the GIM has been carefully selected so as to provide a minimum of bother to the GIM user with regard to GIOC idiosyncracies of absolute addresses, protection, appending and segmentation, etc. At the same time, the list interface allows the DIM writer a maximum flexibility in operating a particular device through suitable symbolic DCWs called pseudo-DCWs. The interface also provides a reasonable way of decoding GIOC hardware status words so as to provide a maximum amount of information to the user. The structure of the list interface will be discussed in the next section; a look at a given list structure now seems in order.

List Structures

Ignoring for the moment the question of where a user may expect to find his lists, it is instructive to consider the format of a given list. Suitable declarations for a DIM writer's list are:

```
/* Declarations for the List Status Tables */

dcl 1 lst based(p),              /* list status structure */
    2 len fixed bin(12),         /* length of list(i) */
    2 tbase fixed bin(12),       /* LPW tally at start of list */
    2 adrlst ptr,                /* pointer to address list(i) */
    2 dcw ptr,                   /* ptr to DCW list (if list is
                                    active) */
    2 pdcw(len),                 /* array of pseudo-DCWs */
      3 optp bit(6),             /* type of pseudo-DCW */
      3 pdcwd bit(84);           /* pseudo-DCW itself */
```

```
/* Address list declarations */

dcl 1 adrlst(len) based(p),    /* optional data address array */
    2 segno bit(18),           /* segment number from addr ptr */
    2 offset bit(18);          /* segment offset from addr ptr */
```

Several items bear further explanation. Consider the item "lst.adrlst". If a pseudo-DCW indicates data transmission to/from the user's area, it will be convenient to indicate both the segment number and the word offset of the start of this data area. Thus, associated with each pseudo-DCW in a list, one may also find it convenient to associate an "address space pair" indicating the area of data transmission. If no pseudo-DCWs will involve data transmission (e.g. a list of CCWs) then no address space is needed. In this case, "lst.adrlst" is null.

By the same token, one may only need DCWs under certain conditions. Therefore, the List Status Table (LST) includes a pointer to the DCWs which are allocated on demand. A null pointer in "lst.dcw" indicates no DCWs are allocated for the list.

Lists and the Class Driving Tables

For each device attached to a Multics system, the GIM will always associate one of a number of Class Driving Tables (CDTs) which relate the characteristics of the device in question. One could conceive of a CDT as an "include" file which contains all the relevant "declarations" for a particular device although the analogy to EPL should not be over-emphasized. Having established a CDT for a particular device, the GIM can always interpret a user's symbolic list of instructions by consulting the CDT for a translation of the symbolic instructions into GIM - or GIOC - understandable actions. Moreover, a suitably defined CDT can also constrain and guide the requests along the paths of correct operation. For instance, material contained within the device description of a CDT for a Model 37 Teletype might very well include the fact that a "seek" operation doesn't exist. Information contained within any given CDT falls into 6 major types, hereafter called "op-types":

A. Five GIOC DCW creation and editing requests

   i) Channel Command Word (CCW) manipulation to facilitate starting and stopping a channel.

ii) Command Data Control Word (CDCW) manipulation to allow device-oriented commands such as rewind, etc.

iii) Transfer Data Control Word (TDCW) manipulation to allow programmed branch of control within the GIOC.

iv) Literal Data Control Word (LDCW) manipulation for sending constants to devices (e.g. marks, spaces, etc.)

v) Data Control Words (DDCW) manipulation to facilitate actual data transmission to/from the GIOC.

B. One GIOC hardware status translation request

vi) Interrupt Return Status

All of the requests in group A above allow the user to issue symbolic editing requests and alter the state of the GIOC-oriented DCWs and CCWs associated with the device in question. On the other hand, the interrupt return status request in group B causes the GIM to take a hardware status word and turn it into a symbolic format. The translation in group B is a "de-compilation" in effect.

To summarize, in group A, the user hands over a list of symbolic edit requests and causes the GIM to edit various hardware words. In group B, the user receives symbolic translations from the GIM derived from hardware words.

Hard-Core I/O System Data Bases

Data bases and structures within the GIM are divided into two rough categories: 1) System-wide data bases, 2) Per-device data bases. The per-device data bases can be readily described as they are encountered. System-wide data bases are used throughout the GIM so a brief preview of their format and function now seems in order.

A. Hard-Core I/O System Static Storage

Many system parameters relevant to GIOC operation throughout the GIM reside in a static storage segment, hcio_stat_. These values are set at I/O system initialization time (see MSPM BL.8.00) and are never altered except for dynamic reconfiguration (e.g. add a GIOC to the system) or system experimentation. The following items reside in hcio_stat_:

```
nadpt(i) fixed bin (17)  /* number of adapters
                            on GIOC(i) attached
                            to system */

nlchnls fixed bin (17)   /* number of logical
                            channels in Multics
                            configuration */

ngiocs fixed bin (17)    /* number of GIOCs attached
                            to system */

n_stats fixed bin (17)   /* highest allowable status
                            channel number for GIOC */

n_phys_chnls(i) fixed bin (17) /* (highest usable
                            physical channel number of
                            any attached GIOC(i))
                            divided by 2 */

cbufsiz fixed bin (17)   /* length of connect channel
                            buffer area */

stat_len fixed bin (17)  /* length of each status
                            channel buffer area */

tdepth fixed bin (17)    /* maximum length of
                            transfer chain */
dctp ptr                 /* pointer to Device
                            Configuration Table
                            (BF.3.10) */
catp ptr                 /* pointer to Channel
                            Assignment Table (See
                            below) */
cstp ptr                 /* pointer to Channel
                            Status Table (See below) */
dcwp ptr                 /* pointer to DCW segment
                            (See below) */
datap ptr                /* pointer to DATA segment
                            (See below) */
```

In addition, hcio_stat_ contains all the equivalences between symbolic errors detected in the GIM (e.g. system or machine error) and the error code returned to the DIM user. A complete catalog of these error codes and other relevant error information may be found in MSPM, BF.20.05.

B. Channel Assignment Table (CAT)

The Channel Assignment Table (CAT) contains information relating each GIOC and device to the Multics configuration. The CAT serves as a general index to most of the GIM data bases and devices and, as such, serves a fundamental role in the operation of the GIM. The declaration for the CAT is related to the exact Multics configuration; a typical configuration is shown below:

```
/* Declarations for Channel Assignment Table */

dcl 1 cat based(p),                     /* assignment table (wired-
                                           down) */
    2 safep ptr,                        /* pointer to canned stop
                                           sequence */
    2 chnary (250 /* nlchnls */),       /* array of channel poop */
      3 padding bit(1),                 /* bit diddling */
      3 devx bit(17),                   /* device index */
      3 lctseg bit(18),                 /* segment no of lct */
    2 gioctab(2 /* ngiocs */),          /* GIOC association array */
      3 base bit(18),                   /* segment number of GIOC
                                           base */
      3 padding bit(18),                /* bit diddling */
      3 slct(0:3 /* n_stats */),        /* lct for stat chans */
        4 segno bit(18),                /* segment number */
        4 off bit(18),                  /* and offset */
      3 clct(0: 2),                     /* lct for connect chans */
        4 segno bit(18),                /* segment number */
        4 off bit(18),                  /* and offset */
      3 adptab(20 /* nadpt(i) */,       /* table of GIOC adapters */
        4 abase bit(18),                /* starting channel number */
        4 nchn bit(12),                 /* number of chan in adpt */
        4 atp bit(6),                   /* type of adapter */
      3 logchn(7: 255 /*
        nphys_chnls(i) */) bit(12); /* logical channel(phys_
                                           channel) */

dcl catp ptr ext static,                /* pointer to CAT */
    nadpt(2) fixed bin(17) ext static,/* number of adapters per
                                           GIOC */
    nlchnls fixed bin(17) ext static,/* number of logical
                                           channels */
    ngiocs fixed bin(17) ext static,/* number of GIOCs */
    n_stats fixed bin(17) ext static,/* highest status channel
                                           number */
    nphys_chnls(2) fixed bin(17) ext
    static;                             /* maximum physical channel
                                           number/2 */
```

The segment name of the Channel Assignment Table (CAT) is "cat_seg".

C. Channel Status Table (CST)

The Channel Status Table (CST) can be considered to be an extension of the GIOC hardware status queues. Hardware status words are transferred from the hardware queues and placed in appropriate slots in the CST, thus eliminating certain interlock problems involved in manipulating the hardware status queues while the GIOC may also be manipulating them. Also, the CST serves as a potentially large and pageable buffer area in contra-distinction to the small and wired-down hardware queues. The name of the segment containing the CST is "cst_seg". A suitable declaration for the CST is:

```
/* Declarations for Channel Status Tables */

dcl 1 cst based(p),                        /* channel status table
                                              (wired) */
    2 lock bit(36),                        /* update and removal
                                              interlock */
    2 xtab(0:  250 /* nlchnls */),         /* index table, where
                                              0 = vacant list */
      3 adper bit(1),                      /* ON if adapter has made
                                              error */
      3 fstx bit(17),                      /* index to first status(i) */
      3 giocer bit(1),                     /* ON if whole GIOC has made
                                              error */
      3 lstx bit(17),                      /* index to last status(i) */
    2 x1 bit(1),                           /* padding */
    2 hix bit(17),                         /* highest index currently
                                              used */
    2 padding bit(18),                     /* bit diddling */
    2 stat(500 /* cst_len */),             /* status frames */
      3 x1 bit(1),                         /* padding */
      3 nxtx bit(17),                      /* index to next status
                                              block */
      3 time bit(54),                      /* time of last interrupt
                                              of channel(i) */
      3 statwd bit(72);                    /* interrupt status word
                                              of channel(i) */

dcl cstp ptr ext static,                   /* pointer to Channel
                                              Status Table */
    cst_len fixed bin(17) ext
    static;                                /* length of status array
                                              in CST */
```

D. GIOC Mailbox Areas

The GIOC mailbox areas are the heart of the GIOC operation. It is within these areas that the GIM reserves complete license for manipulation. The implementation of the GIM requires one distinct mailbox segment for each GIOC attached in the Multics configuration. Initial implementation of the GIM has given the name "gioc_mbxN" to these segments. N, as one might guess, covers the range 1, 2, ..., "ngiocs". Suitable declarations for the GIOC mailbox areas are:

```
/* Declaration for GIOC mail boxes */

dcl 1 mailbox based(p),                 /* p points to the base of
                                           GIOC mailboxes */
    2 scw(0: 7),                        /* the 8 status control
                                           words */
      3 scwa bit(36),                   /* word a */
      3 scwb bit(36),                   /* word b, the active one */
    2 cpw(0: 2) bit(72),                /* connect channel pointer
                                           words */
    2 x1(3) bit(72),                    /* 3 unused boxes */
    2 data(7: 2047),                    /* data channels */
      3 lpw bit(72),                    /* list pointer word */
      3 dcw bit(72);                    /* DCW mailbox */
```

E. GIOC DCW Area

Since the GIOC does not have a counterpart to the 645 processor appending hardware, the DCWs necessary for driving the devices attached to a GIOC must reside in core in a manner reminiscent of absolute programs. That is, the DCW area must be wired-down segment that is either unpaged or paged contiguously. The GIM will be responsible for allocating and freeing space within the area so that a user's DCWs may be placed in the area and released after they are used or are no longer needed. The segment name for this segment is "dcw_seg".

F. GIOC Data Area

In the same spirit as the GIOC DCW area, a wired-down segment, "data_seg", is needed for a user's data buffer area. On all I/O instructions involving writing, the user's write buffer is first copied by the GIM into "data_seg". Subsequent I/O via the GIOC will write the data onto the appropriate device. Similarly on calls involving reading, the GIOC will initially place the data in "data_seg". Subsequent actions of the GIM will copy the data into the DIM user's buffer area.

One might raise the objection that the above description of I/O implies that all Multics user I/O will always require at least one move. This is true.

A counter-argument hinges on the fact that one is attempting to perform I/O into pageable, non wired-down areas with a device that recognizes neither paging and page boundaries nor wired/unwired areas. A moments thought will quickly reveal the inherent difficulties and incompatibilities between the two ideas. Moreover, it seems clear that the extra buffer area is the best way to resolve the existing incompatibilities.

GIOC Channel Activity - lpw$safe, lpw$active

One of the most perplexing design problems centered around a method for determining when a channel was or was not active. Unfortunately, the design of the GIOC does not provide any infallible and simple method for inquiring as to a channel's status. One could set switches whenever the channel was started and reset the switches when a channel termination was detected. Such a method could be made to work although it is apparent that much synchronization and general bookkeeping in a system-wide data base would be necessary.

A much simpler (and presumably fail-safe) method has been adopted by the GIM. One rule is adhered to:

"A channel is active unless proven otherwise".

Fortunately, a simple method of proving inactivity exists. During system initialization, two DCWs are generated and stored in the DCW segment, dcw_seg. The first DCW is a command DCW (DCW type 4) with the "last-DCW-bit", bit 17, set ON. Whenever any data channel processes this DCW, it immediately terminates activity. The second DCW is placed immediately adjacent to the first DCW and is a transfer DCW. It transfers to the first DCW. Thus, if a data channel processes either DCW, it will quickly cease activity. Should the channel accidentally get bumped off of the command DCW, it will only return via the transfer DCW.

The test for channel activity quickly resolves into testing the List Pointer Word (LPW) of a channel to see if the LPW is pointing at either of the two "safety" DCWs.

In fact, the call to lpw$active simply sets the activity bit by a PL/I logical statement to the effect that the LPW is or is not pointing at the DCWs indicated as "safety" DCWs in the Channel Assignment Table (CAT) at the entry "cat.safep". The LPW is set to the safety DCWs by one of 4 means:

1. The channel is initialized as such during system initialization.
2. The end of a DCW list was reached and no user provision for stopping the GIOC was encountered.
3. On a call to request$status made by a DIM, the GIM noticed a hardware status indicating channel termination.
4. The GIM stopped the channel in response to a DIM call.

The GIM may elect to stop a channel and require that it cannot be accidentally restarted. The internal call lpw$safe is used in such cases.

To safely stop the channel, the List Pointer Word (LPW) for the indicated channel is set to point to the command DCW of the "safety" pair. The data channel mailbox is filled with the command DCW of the "safety" pair. Finally, both mailboxes are matched against what was just placed in them. Transient or intervening channel activity may have overwritten the "safety" DCWs so the shutdown did not have the proper affect. If the mailboxes do not match the original "safety" DCWs the above insertions are repeated until the mailboxes do match the safety DCWs. The channel is now inactive and will stay inactive until positive action is taken to restart it. Note that the above-mentioned "safety" action may not be noticed immediately by a GIOC direct channel since this type of channel only refers to its mailboxes when a DCW is needed. That is, a direct channel will only terminate upon completion of processing a DCW. This may delay a direct channel shutdown by some small period of time but the shutdown is inexorable.