DRAFT: 6/19/68
C. T. CLINGEN

[redacted] final?

See further

Overview of Multics I/O System

Purpose

Key Design Features

The objectives of this design are to achieve an I/O System which is parsimonious in its requirements for space and event channels, simple in structure and highly efficient when performing those I/O requests most frequently required. At the same time the design must be adequate to support several types of devices, including TTY37's, 1050's 2741's, and ultimately magnetic tapes, and disc files, permit proper I/O processing in the event of quits, saves, resumes and restarts and operate either entirely within a single process or as part of both a working process and a Universal Device Manager Process (UDMP).

love motherhood

packs? cards punch-port

Below is a list of some of the salient features which characterize a design which satisfies there requirements.

1) There will be a simple high-speed I/O switch which is essentially a transfer vector and takes advantage of some of the facilities provided by the standard Multics linkage mechanism. Appropriate linkage sections are grown during the attachment operation for subsequent use during I/O calls.

2) Device attachment will be per process rather than per process group.

3) Each Device Control Module (DCM) -- there will be one for each c of device -- will be driven both from "above" by user I/O requests or from "below" by GIM interrupts by means of a single event call

Motivation (save, resume, selective restart) each process basically independent)

channel. Device identification is achieved by passing the DCM a ~~relative~~ pointer argument in the event call channel which pointer locates a unique block of per device data in the Process Driving Table (PDT) available to each DCM.

4) All the volatile data and buffer areas required to operate each device will be contained in a single fixed-length data base, one per device attachment, called a Device Data Base (DDB).

5) I/O calls to be implemented initially include: attach, detach, read, write, abort, quit, hangup, logout, restart, make_ddb (initial divert), reset_ddb (revert and subsequent diverts). Subsequent to the implementation of these calls, the following calls will be provided:

6) Targeted performance objectives of the design are:

   a) No terminal device should require more than eight(8) 1K pages of pure code and invariant data for its operations during normal I/O read and write operations.

   b) No terminal should require more than four(4) 1K pages of per-process data, including stack frames, during normal I/O read and write operations.

   c) Average time to read or write a line of I/O for a remote console should not exceed .1 ms. of cpu time (exclusive of page, segment and wall-crossing faults) per character processed if no data

Percent overhead ~ for a 25 char line, lets say

conversion is required and .2 ms. if data conversion is required.

Major I/O System Components

The major procedures and data bases comprising the I/O System are represented in Figure 1. The general function of each of these components is briefly summarized below.

User procedures are allowed to make read and write calls to the I/O system once an attach call has been issued associating an I/O name with a particular I/O system procedure name and specifying the device being attached. Similarly, a user can also detach a device from the specified I/O name and subsequently attach a different device if he so desires.

The attach procedure has two principal functions. First, it constructs, using the arguments of the attach call, a switch entry using the Multics linkage section formats, so that subsequent user read and write calls are routed directly to the outer module specified in the attach. In Figure 1, the outer module attached is the Device Strategy Module (DSM) as will generally be the case. Secondly, the attach procedure obtains a fresh copy of the DSM linkage section (for each process there is one copy per attached device) and ~~initializes it with~~ a pointer to ~~a copy of an~~ empty Device Data Base.

The working process attachment module is consulted when a user requests the attachment of a particular device or class of device, or when the user requests that a device be detached.

The DCM attachment module is invoked when a quit or hangup has occurred for the device, so that the device can be properly released. Also the DCM attachment module is invoked in the initialization dialogue between the user process and the UDMP which sets up those conventions required for rapid interprocess communication for the asynchronous processing of read and write requests.

Both attachment modules call the I/O Attachment Module (IOAM) in ring 0 to obtain confirmation that the required device assignment changes are valid. When device reservation is implemented in Multics, the IOAM will be the central control point for the reservation and distribution of devices.

The Device Strategy Module (DSM) in a user working process and the Device Control Module (DCM) normally in the Universal Device Manager Process form the heart of the I/O System. The DSM takes data ~~implied~~ by user write requests and places it in a write-behind buffer in the shared DDB assigned to the attached device for subsequent conversion and transmission by the DCM. The DCM takes ~~output~~ from the device and converts it, placing it in a read-ahead buffer in the DDB for subsequent consumption by the DSM in response to a user read request. (Note: perhaps both kinds of buffers need never co-exist in the same DDB.) All necessary synchronization between DSM and DCM is accomplished by simple Interprocess Communication signals and cues placed for interpretation in the DDB.

A Process Driving Table (PDT) resides in each UDMP. It contains per-device information in fixed-length PDT blocks, ~~the index~~ to which is used to communicate to the DCM the particular device involved in each I/O transaction.

The DCM is event-driven with a single event call channel used both by all user processes requiring I/O in the UDMP and by the Device Signal Table Manager as a result of termination and special interrupts received by the GIM for active devices. Device identification is achieved by ~~passing~~ associating the ~~index~~ pth to the appropriate PDT block ~~as an argument to the~~ with the call event.

Data Base Layouts

The key data bases in this design are the Device Data Block and the Process Driving Table. Each attachment creates a Device Data Block for the attached device. In addition to a header, the DDB contains an Interprocess Communication Block (ICB), a Device Status Block, some buffers and a DCM work area (see Figure 3).

The ICB contains information needed for interprocess communication, key of which is the event call channel over which the DSM can signal UDMP when it services are needed and the PDT block index corresponding to the device using this DDB. also DSM evchn when needed

The Device Status Block contains buffer synchronization information relevant to the proper interlocking of DCM-DSM read-ahead and write-behind processing.

Also, each DDB contains a circular Read Ahead Buffer and Write Behind Buffer, each of fixed length, for shared use by both DCM and DSM. Each buffer contains a data block, each prefixed by a transaction status block describing the exact status of the associated data block -- e.g., error recovery information. Transaction status blocks are forward threaded permitting variable length data blocks to be used.

Finally, there is a DCM work area into which data is placed for subsequent code conversion and canonicalization as required.

The Process Driving Table (see Figure 3) contains fixed sized blocks, each containing information useful to the DCM for identifying particular devices and the process and DDB which correspond to the device. Most interesting entries in each PDT block are the pointer to the DDB belonging to the device identified by the device index, the process id of the device assigner, and the id of the event channel over which to signal the working process when the UDMP has caught up and can once again accept I/O requests from the working process' DSM.

Since searches through the PDT are required when devices are reassigned to different processes -- as the case for quits, for example, these searches must be rapid. This suggests that PDT blocks be sorted alphabetically by Device ID and that a binary search be used to locate a desired block in the PDT.

The Switch

The purpose of the switch is to permit the user to declare attachments between specified I/O names and I/O System modules. Subsequent references to the I/O name are directed to the associated I/O module.

The attach call is of the form:

call attach ("a"/* io_name */, dsm /* I/O system module */, etc)

with subsequent I/O calls of the form:

call io$write_a (etc.).

The switch proper is comprised of an impure procedure named io and its linkage section io.link. The attach procedure, when invoked, first allocates space in io so that an external symbol definition plus three additional words can be placed in io for each outer I/O call (e.g., io$read_a, io$write_a) permitted to the DSM as a result of this attachment. Similarly, space for entries in io.link is also allocated.. Then link_change$make_definitions or equivalent is called to create an external symbol definition in io for each outer I/O call, ~~creating at the same time a trap before definition for each external symbol definition. Also the entry point for each outer call must be initialized with~~ an "eaplp" ~~and a "tra" at this~~ time. The switch is now initialized for the first outer call to, say, io$write_a.

When the first call to io$write_a occurs, ~~a trap before definition will~~ be sprung. The trap procedure invoked will then store in io a tra * + 1 followed by an its pair to the appropriate location in dcm.link (presumably taking another linkage fault to do so) -- see Figure 4. The trap procedure then returns, the linker completes its link from the user to the io$write_a entry in io.link and the switch has been initialized. All subsequent references to io$write_a will result in a transfer to the entry dcm$write in dcm.link as desired.

Note that if the trap procedure is expected to build an its pair to some outer module other than dcm, the attach mechanism will need to keep a per process table of attachment pairs relating I/O names to outer I/O modules so that the trap procedure is able to look up the desired correspondence between I/O module and I/O name.

The Attach Call

The key steps in the attach logic ~~is~~ are summarized below. The exaple described is:

call attach (a, dsm, ~~device~~ type, etc.).

Consult Figure 1, 2 and 3 as needed.

Working Process

1. Determine that the user has access to the requested device type. (Call working process attachment module which calls IOAM).
2. Build a switch entry to dcm$read, dcm$write, etc. (see above).
3. Obtain a new copy of dcm.link. Create a new Device Data Base, in the process directory, for the newly attached device. In internal static of dcm.link place a pointer to the new DDB. (Note: ~~dcm~~.link will be a gate segment into ring 2).
4. Create an event wait channel for this device for use by the UDMP.
5. Initialize the DDB:
   - Initialize the header (this may be device dependent).
   - Place the Device Name (this is an argument in the attach call) in the ICB.
   - Place the Working Process Event Wait Channel id in the ICB.
   - Initialize the remainder of the DDB.
6. Signal the UDMP over its special initialization event call channel passing as an argument the unique bit string from which the unique name of the DDB was constructed.
7. Call wait.

Universal Device Manager Process

Upon being awakened over the special initialization event call channel, the DCM attachment module in the UDMP will perform the following initialization steps.

1. Using the unique bit argument of the DDB belonging to the working process and the working process id, make the DDB known to the UDMP.
2. Locate the PDT block for the device named in the ICB of the DDB. Initialize the PDT block with the user id, segment number (in UDMP address space) of the DDB (from step 1 above), working process id, and the working process event channel id (from the ICB of the DDB). leave latter in ICB

   Other items are assumed set at system initialization time.
3. Initialize the DDB with the UDMP event call channel name and the PDT block index.
4. Wake up the working process. The working process now awakens in the working process attachment module and realizes that initialization is complete.
5. Call wait.

why just create a special event channel for this device? then you don't need to do polled activity DCM

Detachment (note DSM called first then detach does its work)

When a device is detached, its copy of dcm.link is discarded along with its DDB. Then all entries for this I/O name in the impure io procedure which contain its pairs to dcm.link are modified to contain pointers to notfounder.link. In this way, subsequent calls to drive the detached device will be intercepted by the notfounder.

should flush pending I/O?

Read/Write Transactions

Two concepts are central to the design of the read/write mechanism of the I/O system: 1) buffer strategies and 2) process synchronization.

The buffer strategy employed is to maintain two circular buffers, one for read-ahead and the other for write-behind. Each buffer can be concurrently used by both DSM and DCM but need never be interlocked. This is made possible by maintaining read-ahead and write-behind pointers for both DSM and DCM (see Figure 2) and by enforcing the obvious rules that ~~the~~ neither read-ahead pointer may ever be allowed to overtake the other read-ahead pointer or/and that neither write-behind pointer never be allowed to overtake the other write-behind pointer. As long as both DSM and DCM are progressing through the buffers in such a fashion that neither is ever roadblocked waiting for the other, both may be allowed to progress in essential ignorance of the other. Normally, however, such is not the case, giving rise to the need for a means for process synchronization. This is achieved by means of interprocess communication.

A straightforward sychronization strategy goes something as follows. There are four cases; in the first two the DSM for a particular working process becomes roadblocked. In the second two, the DCM becomes roadblocked for a particular terminal.

1) The DSM read-ahead pointer catches up to the DCM read-ahead pointer.

   a) Set the DSM waiting for read ahead switch in the DDB for this device.

b) Send a call event to the DCM using the UDMP event call channel specified in the ICB of the DDB. (Use the PDT block index as the "argument").

c) Call wait.

2) The DSM write-behind pointer catches up to the DCM write-behind pointer.

a) Set the DSM waiting for write behind switch.

b) Send a call event to the DCM. (Use PDT block index as the "argument").

c) Call wait.

3) The DCM read-ahead pointer catches up to the DSM read-ahead pointer.

a) Set the DCM waiting for read ahead switch.

b) Send a wakeup to the Working Process using the process id and event wait channel id found in the PDT block for this device.

c) Call wait (to see if there are any other outstanding call events).

4) The DCM write-behind pointer catches up to the DSM write-behind pointer.

a) Set the DCM waiting for write behind switch.

b) Send a wakeup to the Working Process.

c) Call wait.

Note that when the DCM awakens an event call channel it checks the DSM waiting for write behind and DSM waiting for read ahead switch to determine whether it should read or write next. The DCM switches may be redundant if this synchronization strategy is used.

Since the DCM may also be awakened by a GIM interrupt, there is a distinct possibility that the DCM will be asked to place information in the read-ahead buffer only to discover that the DSM has as yet been unable to awaken and empty it. Therefore, a provision must be made to halt further input until buffer space once again becomes available.

Given this somewhat oversimplified description for buffer handling and process synchronization and given Figure 1, the steps to be taken to perform for input and output transactions are relatively self-evident.

crude balance possible if no DCW's made for read if read able space tight

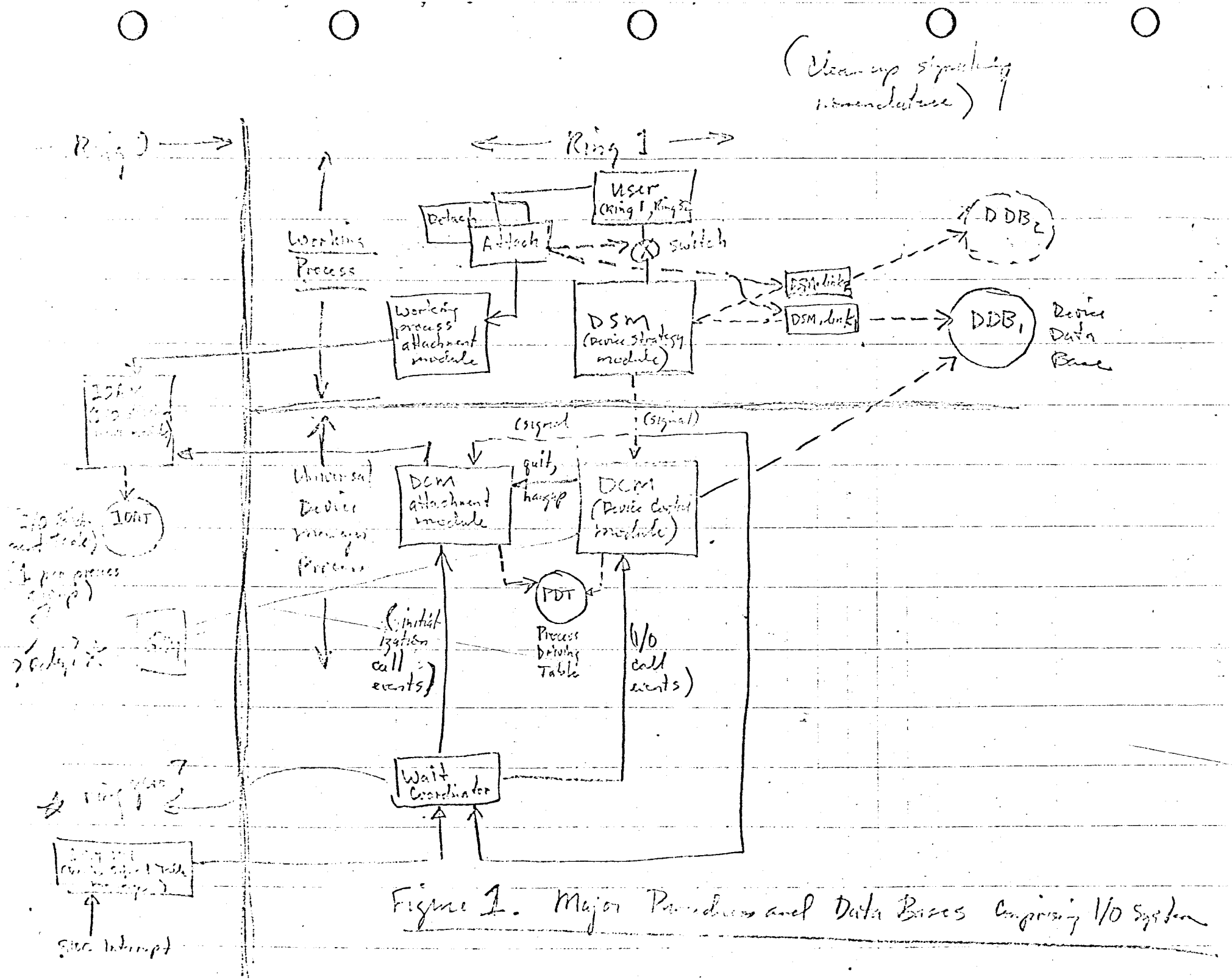

Figure 1. Major Procedures and Data Bases Comprising I/O System

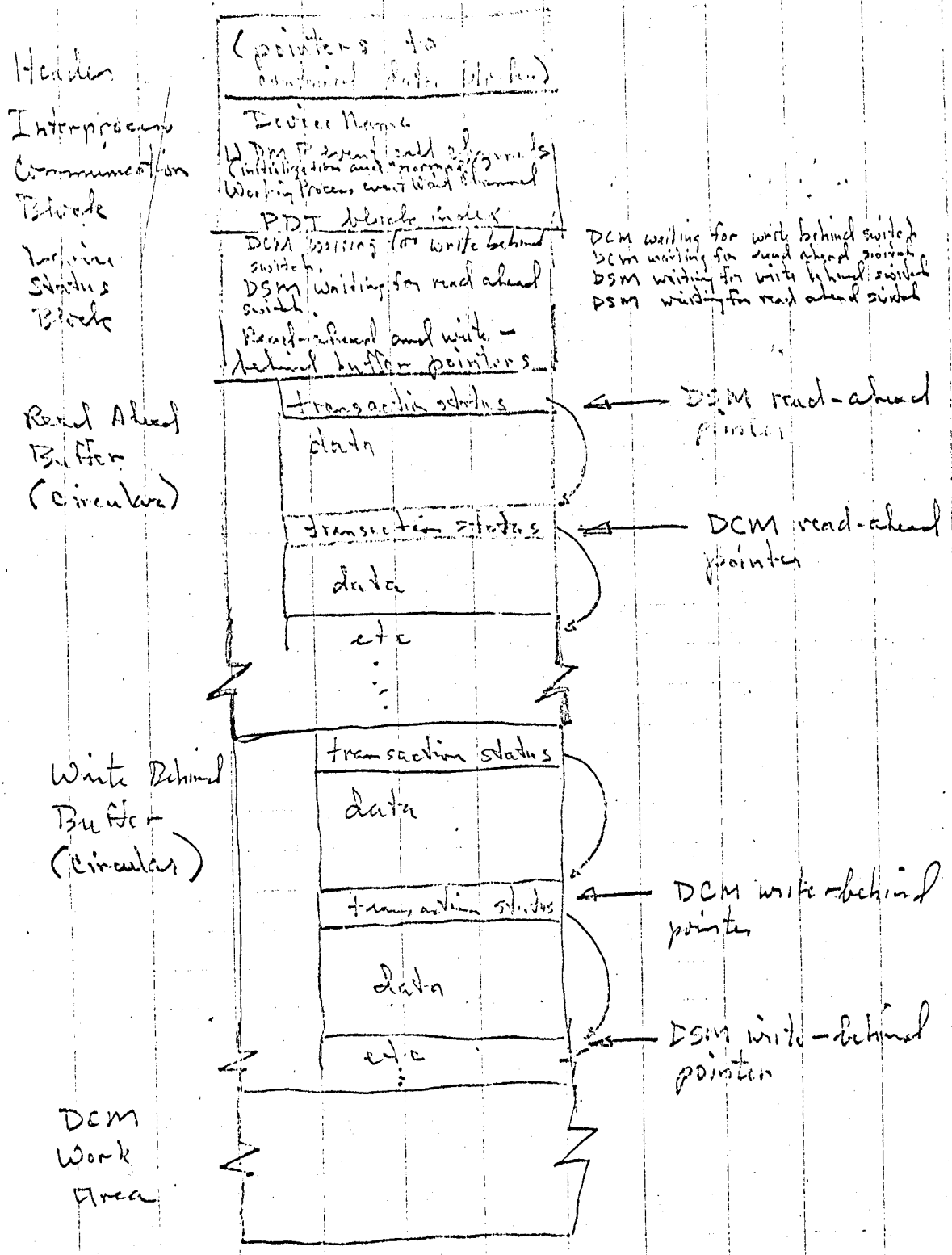

Figure 2 Device Data Base

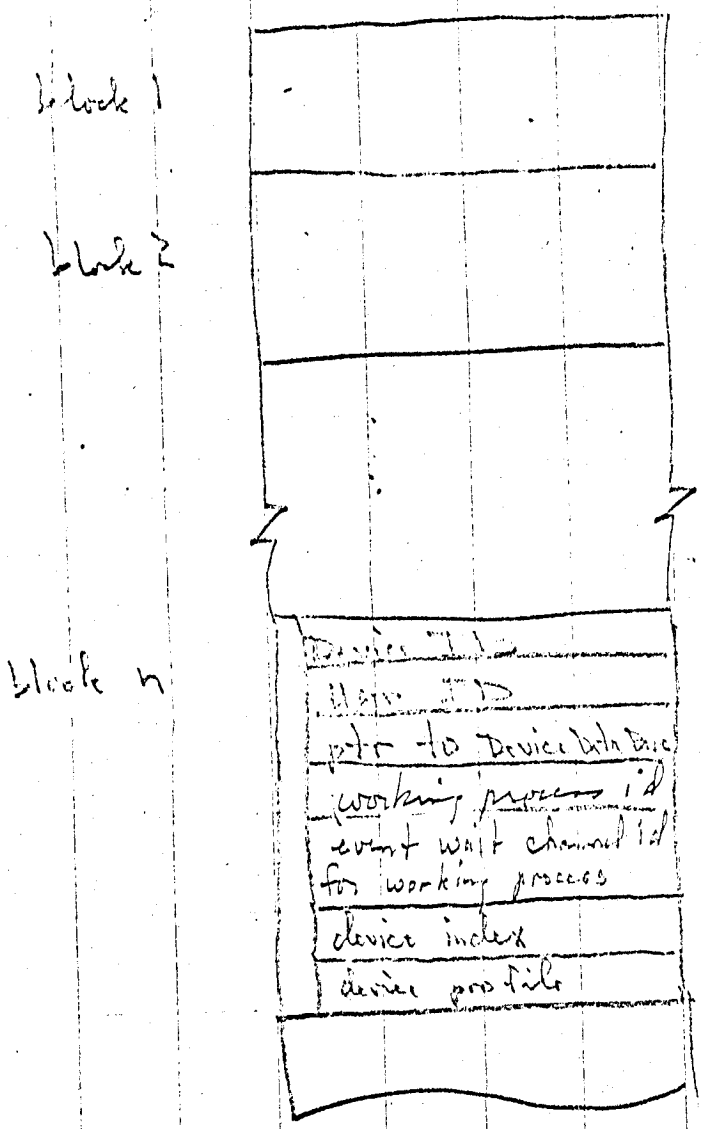

Figure 3. Process Driving Table

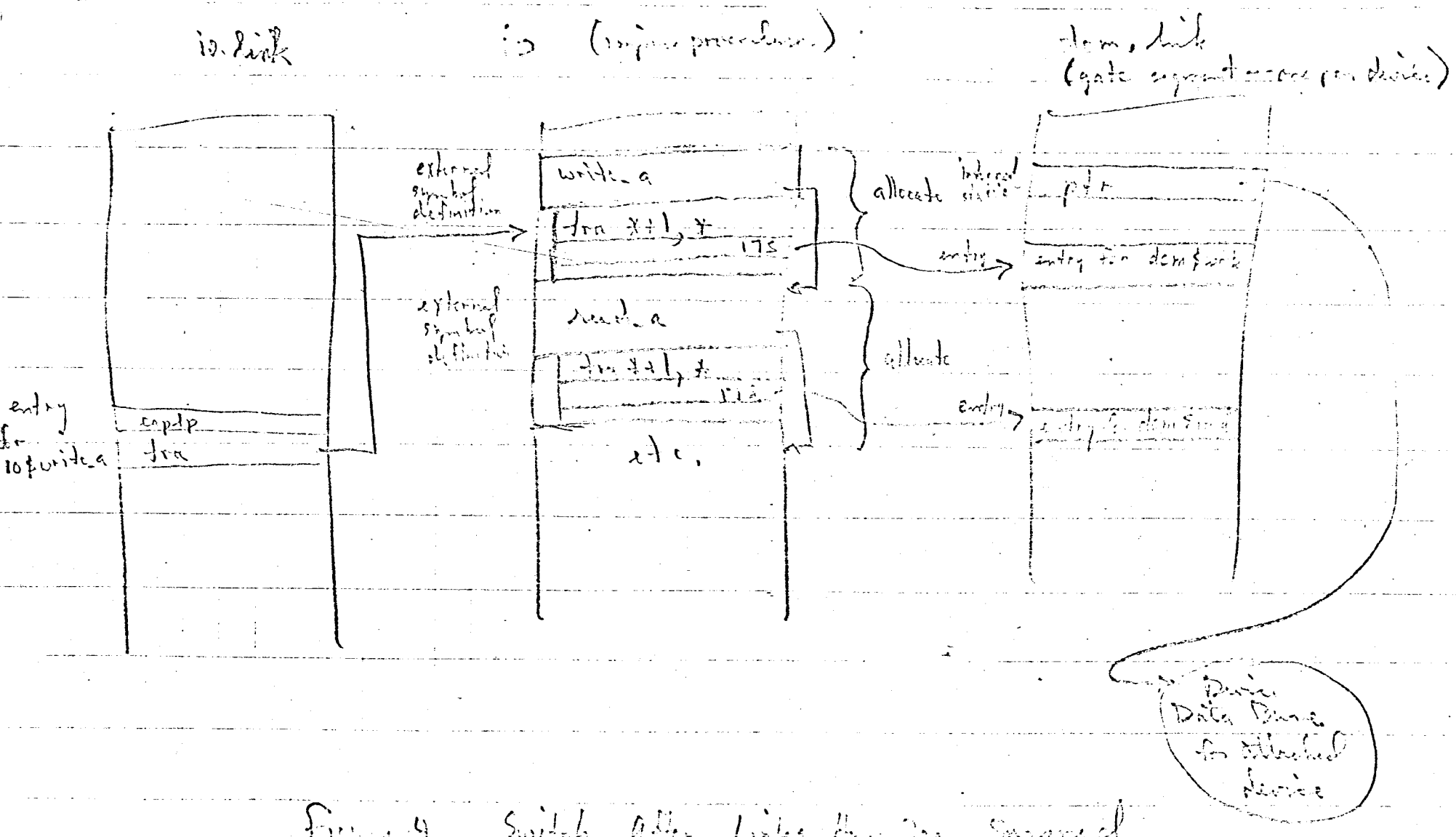

Figure 9. Switch After Links Have Been Snapped