Published: 10/05/67

Identification

Event Channel Manager Primitives and Channel Access Techniques
Michael D. Schroeder

Purpose

This MSPM section describes the methods used by the Event Channel Manager (ECM) to maintain and access event channels. The ECM's associative memory is defined, as is the "hashing" algorithm which operates on the Event Channel Table (ECT). Primitives for creating, retrieving, and destroying event channels and Working Queue (WRKQ) cells are also described. This section will be primarily of interest to system programmers who are maintaining or modifying the Event Channel Manager. It is important to note that the ring 0 version of the ECM, the IPGECM, does not use the same primitives and access techniques. Important differences are explained in BQ.6.05.

Introduction

Familiarity with MSPM section BQ.6.00 through BQ.6.08 is assumed.

As indicated in section BQ.6.00, a sending process knows the target event channel by two symbolic labels: the process id and the event channel name. Using these, the sending process must locate the ECT segment of the receiving process, and then find the indicated event channel within that segment.

The Segment Management Module primitive initiate (BD.3.02) is called to convert a path name constructed from the symbolic process id to a pointer to the proper ECT segment. Because the ECT segment is always known by the symbolic name "ect" within any process directory, the path name

"root>pdirdir>[process id]>ect"

uniquely identifies the ECT segment within the Multics file system. (Note that process id is the only variant in this path name.) The ECT segment identified by the path name is "made known" to the sending process, and a pointer to the base of the segment is returned. The receiving process' ECT may now be accessed directly with this pointer.

The remaining task is to locate the proper event channel within the found ECT. This is done by searching that ECT for a matching event channel name. The search is expedited by the use of a "hashing" algorithm. The search procedure is part of the sending process' ECM, but the data base searched is the actual ECT of the receiving process. Figure 1 in the appendix summarizes this access technique.

As a result of making the receiving process' ECT segment known to the sending process, an entry is made in the Known Segment Table (KST) of the sending process. A process needing to signal a large number of other processes would find its KST growing very large. Some form of housekeeping is necessary to make-unknown ECT segments that are no longer needed. The simplest method would be to call the primitive terminate (BD.3.02) immediately following each access to an event channel. This would remove the entry for the corresponding ECT segment from the KST. The cost of the solution would be high, however, because each event channel access would cause the file system hierarchy to be searched for the needed ECT segment. Instead, a process associative memory (PAM) is defined. The PAM is structured as a threaded list of entries. Each entry remembers a process id and an ITS-pair pointer to that process' ECT segment. The most recently referenced process' PAM entry is always linked to the head of the list. (It is most important to remember that a PAM is associated with each process, and contains entries for other processes to which the containing process is signaling events.) The ITS-pair pointer is valid only as long as the corresponding ECT segment is "known" to the sending process. Thus, ECT segments are made unknown by calling terminate only when their PAM entry falls out the bottom of the PAM. Thus, the number of ECT segments known to a sending process at any one time can be no greater than the capacity of the PAM.

With the addition of the PAM, event channel access now involves the following steps. The PAM is searched from head to tail, following the thread, for a matching process id. If a match is found, the matched PAM entry is relinked to the head of the PAM and the ECT segment pointer is returned. If no match is found, initiate is called to produce a valid pointer, and make known the proper ECT segment. A new PAM entry is created, physically replacing the PAM entry at the tail of the list. This new entry is then logically relinked to the head of the list. Terminate is called for the ECT segment whose PAM entry was replaced. Once a valid ECT segment pointer is obtained, that segment is "hash" searched for the needed event channel. Figure II in the appendix illustrates this access technique.

The "Hash" Coded Search Algorithm

In order to implement the "hash" search, each process includes a separate segment named the Event Channel Hash Table (ECHT). The ECHT is a single dimensional array of ECT indexes. The length of the ECHT is always an integer power of 2. (length = 2^n, n = 1, 2, 3...).

To see clearly how the "hash" search functions, an event channel must first be created in the ECT segment. (A process may create an event channel only in its own ECT.) The ECT is initiated as a forward linked chain of free entries. The index of the head of the chain is in the ECT header. Physical storage for the new entry is simply found as the head of the chain of free entries. (Deleted entries are replaced at the head of the chain, also.) The function, unique bits (BY.15.01), is used to generate the new event channel's name, and the name is set in the entry. The low order bits of the 70 bit name are the clock reading. For small n, then, the low order n-bits will cycle very rapidly, and, on random sampling, will be uncorrelated. Since n-bits will also completely index a table of length = 2^n, the hashing algorithm for an ECHT of that length is:

```
start_index = fixed(substr(ev_chn_name, 71-n,n)) + 1;
```

In the first free ECHT entry after the ECHT entry indexed by start_index is stored the ECT index to the actual event channel. The event channel name is also stored here. The ECHT is the only table actually hashed.

To retrieve the event channel, the sending process must first locate the ECT segment containing the event channel using the previously described method. It must also locate the ECHT segment of the receiving process. (To facilitate this, the PAM in the sending process also remembers the ECHT segment pointer. ECHT segments are made-known and made-unknown in a sending process whenever the paired ECT segment is made known or unknown.) The following steps are then taken.

1. Obtain start index from ev chn name.

2. Compare ev chn name to the channel name stored in ECHT entry (start index)

3. If they match, the index in ECHT entry(start index) is the index to actual event channel in the ECT.

4. If no match occurs, repeat steps 2 and 3 using start index = start index + 1, until a match occurs, the ECHT is exhausted, or an empty ECHT entry is found.

Both the ECHT and the ECT can be extended to maximum Multics segment size, for they each reside in an open-ended segment. When the ECT free chain becomes empty, more ECT entries are added and linked to the free chain. If the ECHT becomes over 3/4 full, its size is doubled and it is rehashed by sequentially searching the ECT for defined Event Channels and "hashing" their names into the ECHT. In a similar manner, when less than 1/4 full, the ECHT may be shrunk by factors of 2. [The ECT cannot shrink, it can only grow.]

Working Queue Cell Access

Event channels may have their capacity expanded by linking Working Queue (WRKQ) cells to them. The WRKQ is also a separate segment within each process. Any sending process may cause an empty WRKQ cell in the receiving process' WRKQ to be appended to an event channel in a receiving process if that channel is in the event queue signaling mode. Since two or more separate processes in separate processors may require this action simultaneously, a race may occur in reserving a free WRKQ cell. For this reason free cells are reserved using the "stac" instruction. The interlock constant is the index of the ECT entry (event channel) to which the WRKQ cell is to be appended.

Free cells in the WRKQ may not be linked together because relinking the free queue would require locking the WRKQ to another sending process. To provide lock-free access, free WRKQ cells are located by a simple linear table search. To keep the search short, the starting index for the search is computed from the event channel index, thus

```
start_index = n* ev_chn_index;
```

where n is the dimension of WRKQ divided by the dimension of ECT, and ev chn index is the ECT index of the corresponding event channel. The search is an "end-around" search. Only if no free WRKQ cell can be found is the size of the WRKQ increased (with corresponding change in the value of n).

Because the WRKQ is also used to provide access control to the ECT, it must be referenced each time a signal is sent to an event channel in the containing process. For this reason, a WRKQ segment pointer is also carried in the PAM entry in the sending process.

The Structure and Operation of the Process Associative Memory

The following declaration defines the PAM:

```
declare  1 pam external static   /*process associative memory*/,
           2 start fixed bin(17)  /*index of "head" of pam*/,
           2 end fixed bin(17)    /*index of "tail" of pam*/,
           2 entry(30)            /*process entry array*/,
             3 prcs_id bit(36)    /*process identification bit
                                    string*/,
             3 1_up  fixed bin(17) /*index of next "higher" pam
                                    entry*/,
             3 ectp pointer       /*pointer to process' ECT
                                    segment*/,
             3 echtp pointer      /*pointer to process' ECHT
                                    segment*/,
             3 wrkqp pointer      /*pointer to process' WRKQ
                                    segment*/,
             3 1_down fixed bin(17) /*index of next "lower"
                                    pam entry*/;
```

The PAM is a single chain of entries. The variable start is the index to the head (or top) entry in the chain, while end is the index to the tail (or bottom) entry of the chain. The two variables, 1 up and 1 down in each entry are the indexes to the next "higher" and the next "lower" entries in the chain, respectively, from the containing entry. Thus, the chain is linked both forward and backward.

The PAM is searched down the chain from entry (start) to entry (end). If a match is discovered, the matching entry is relinked to the top of the chain. If no match occurs, initiate is called three times to locate the needed pointers, and a new PAM entry created. This new entry physically replaces entry (end), and is then relinked to become entry (start). [The ECT, ECHT, and WRKQ segments for the entry deleted are made-unknown.] Figures IIIa and IIIb show the physical and logical structure of a 5 entry AMT. Figures IVa and IVb show the physical and logical structures after entry(4) is referenced. Figures Va and Vb show the physical and logical structure after a new entry is added to the memory in Figures IIIa and IIIb.

Event Channel Deletion

A special problem is encountered when deleting an event channel. It is possible that the event channel to be deleted will be busy at the time the containing process wishes to delete it. The ECM procedures for deleting channels, delet ev chn, checks to see if the channel is

busy. If it is not busy, delet ev chn locks the channel to all users, and deletes it immediately. If the channel is busy, a special lock is set in the channel which allows the process(es) using the channel to finish, but no more to begin using it.

At this point, the deleting process could call "block" and wait for the channel to become quiet. This would not be appropriate, however, because it would force the process to wait for a channel it no longer wants or needs. The busy channel is instead placed in a queue of channels pending physical deletion in the ECT. Later, in a frequently used section of ECM procedure, this queue is checked, and those channels on the queue that are no longer busy are physically deleted. Logically, as it appears to the containing process, the channel is completely deleted when delet ev chn is called. The sometime delayed physical removal of the channel is completely transparent to the user of IPC.

The Event Channel Manager Primitives

The ECM primitives do the actual work of event channel access, including searching and updating the PAM. In addition, they are responsible for maintaining the ECT, ECHT, and WRKQ.

Event channels are created by cre chn. An event channel may be created only within a process' own ECT. Event channels are retrieved from the contained ECT by ret chn. The more involved task of accessing an event channel in another process' ECT is performed by get chn. Channels are erased from the contained ECT by erase chn, which calls upon del chn to perform a subfunction. The pointer to a receiving ECT may be obtained separately with the primitive get tbl.

The working queue is manipulated by three primitives. Cre cell and del cell create and delete cells from a process' own WRKQ. Get cell is used to create WRKQ cells in another process' WRKQ.

It is emphasized that these primitives are an internal part of the Event Channel Manager and, as such, are intended to be called only by the Event Channel Manager itself.

The calling sequences and the argument declarations for the primitives follow:

```
1.  call ecmprm$cre_chn(ev_chn_name,ecex,ecep,sts)
2.  call ecmprm$ret_chn(ev_chn_name,ecex,ecep,sts)
3.  call ecmprm$del_chn(ecex,sts)
4.  call ecmprm$erase_chn(ev_chn_name,sts)
5.  call ecmprm$get_chn(ev_chn_name,ecex,ecep,sts,prcs_id,
         ectp,wrqp)
6.  call ecmprm$get_tbl(prcs_id,ectp,wrqp,sts)
7.  call ecmprm$cre_cell(ecex,cell_type,wqcx,wqcp,sts)
8.  call ecmprm$del_cell(wqcx,sts)
9.  call ecmprm$get_cell(ecex,cell_type,wqcp,sts,prcs_id)
```

```
declare

  cell_type fixed bin(17)    /* 1 = event queue cell
                                2 = associated procedure cell
                                3 = channel access cell */,
  ecep pointer               /* event channel entry pointer */,
  ecex fixed bin(17)         /* event channel entry index */,
  ectp pointer               /* ECT segment pointer */,
  ev_chn_name bit(70)        /* event channel name */,
  prcs_id bit(36)            /* process id */,
  sts bit(36)                /* error status word */,
  wrqp pointer               /* WRKQ segment pointer */,
  wqcp pointer               /* WRKQ cell pointer */,
  wqcx fixed bin(17)         /* WRKQ cell index */;
```

Following is a brief description of each primitive. Error returns from all primitives are indicated by a non-zero value in sts.

1. cre_chn "create event channel entry in own ECT"

   No input arguments are given. An event channel name is created and returned in <u>ev chn name</u>. A free ECT entry is initialized as this channel, and its index and pointer returned in <u>ecex</u> and <u>ecep</u>.

2. ret_chn "retrieve event channel from own ECT"

   <u>Ev chn name</u> is given as an argument. The channel is located using the hash search of the ECT. Its index and pointer are returned as <u>ecex</u> and <u>ecep</u>.

3. del_chn "delete event channel from own ECT"

   The event channel index is given as an argument (ecex). The indexed entry in the process' own ECT is zeroed out and reattached to the queue of free ECT entries. The corresponding ECHT entry is also removed.

4. erase_chn "erase event channel from own ECT"

   The event channel named by ev chn name is erased from the process' own ECT. All connected WRKQ cells are removed and del chn is called to zero out the entry. If the channel is busy an error return occurs.

5. get_chn "get event channel from any ECT"

   The event channel, ev chn name, within the process, prcs id, is located. Prcs id may indicate the caller's process. Ectp returns a pointer to the containing ECT; and ecex and ecep return an index and a pointer to the proper event channel entry within that ECT. Wqcp returns a pointer to the WRKQ in the accessed process.

6. get_tbl "get ECT segment of another process"

   The ECT segment pointer and the WRKQ segment pointers are returned in ectp and wrqp. Prcs id is input to indicate which process' tables are desired.

7. cre_cell "create a WRKQ cell within your own WRKQ"

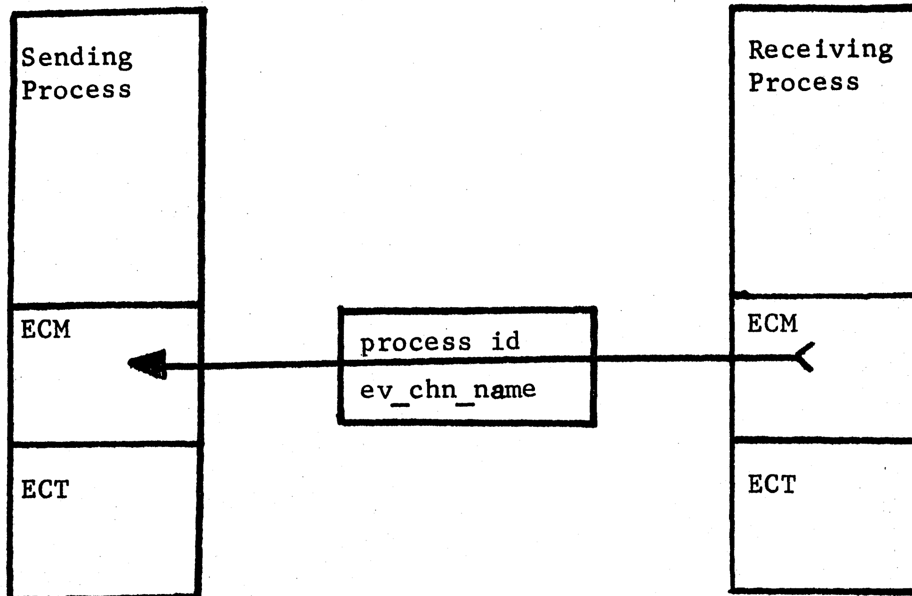
   Ecex is input as an event channel entry index, and cell type indicates the type of cell desired. The WRKQ is searched for a free cell, and the cell is reserved for the channel indexed by ecex. The cell type is set. The WRKQ cell reserved is not linked to the event channel indicated. Wqcx and wqcp return an index and a pointer to the WRKQ cell reserved.

8. del_cell "delete WRKQ cell from own WRKQ"

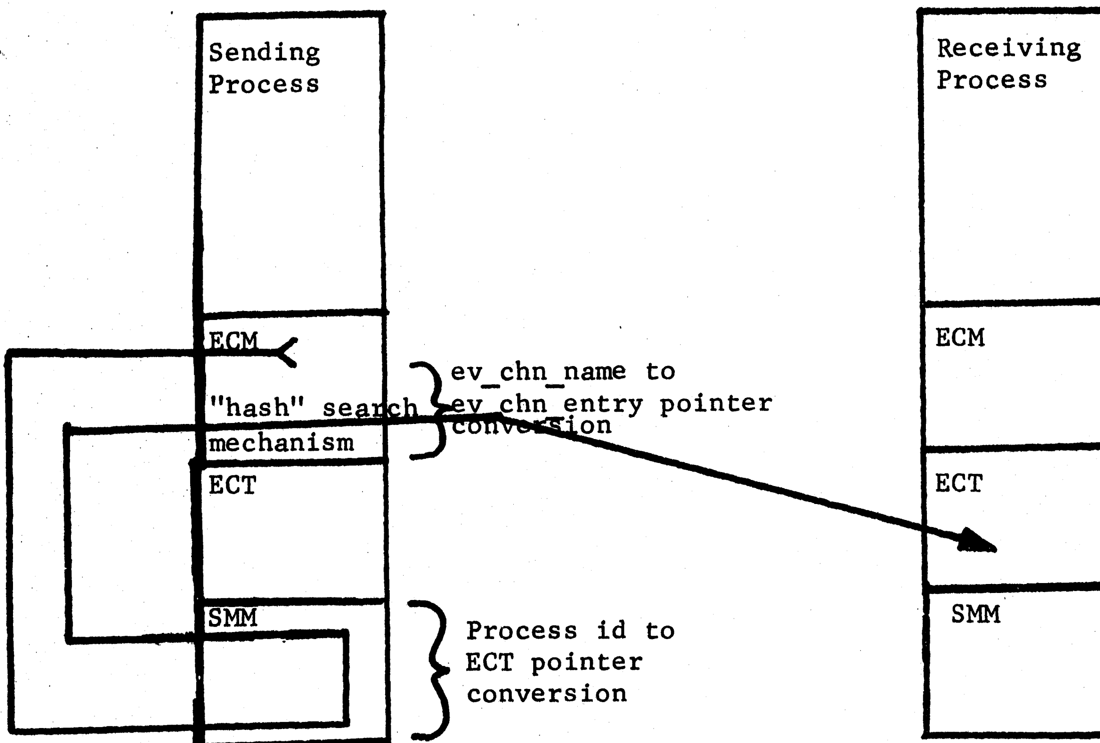
   The WRKQ cell indexed by wqcx is zeroed and returned to the group of empty WRKQ entries.

9. get_cell "create a WRKQ cell in any process' WRKQ"

   Get cell performs the same function as cre cell, but is not restricted to the caller's WRKQ. Prcs id indicates the desired WRKQ.



Basic Interprocess Communication



Channel Access

Figure I

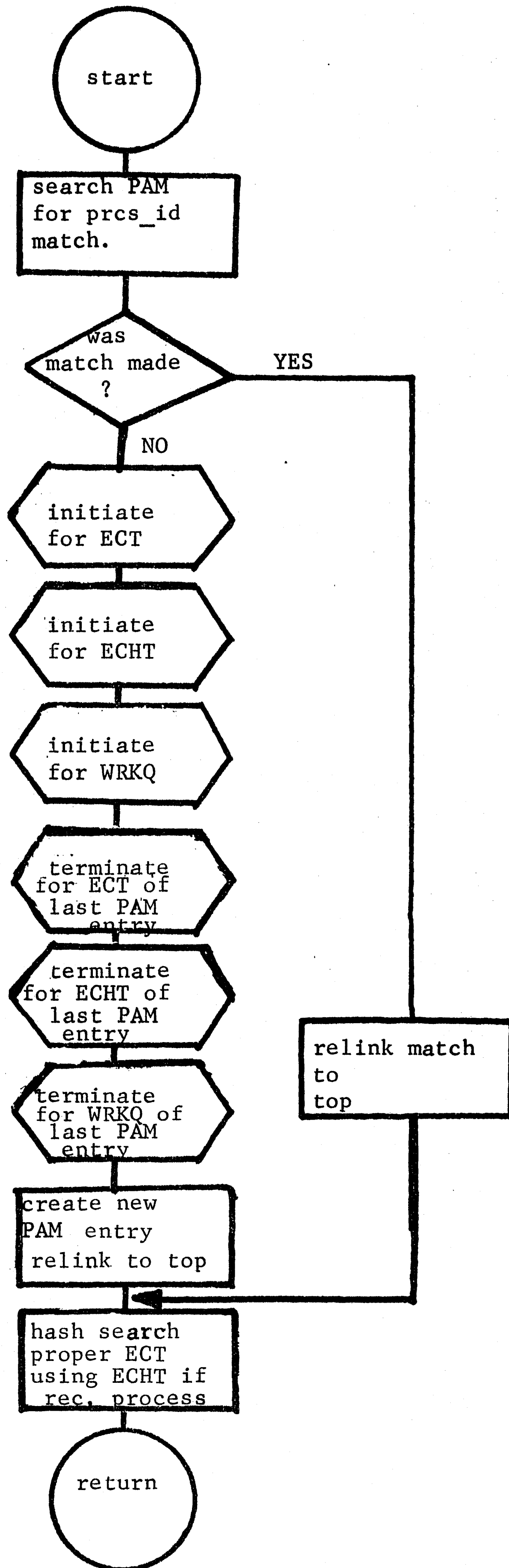
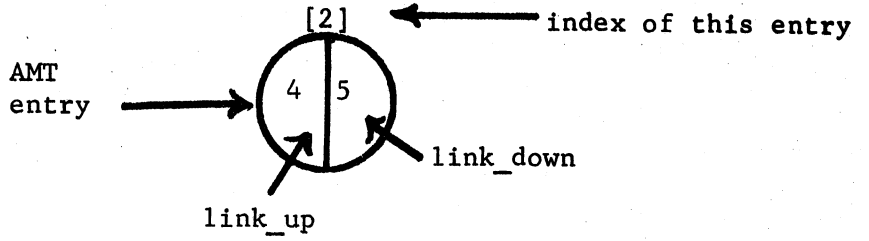


Figure II Channel Access

KEY:



(physical) a

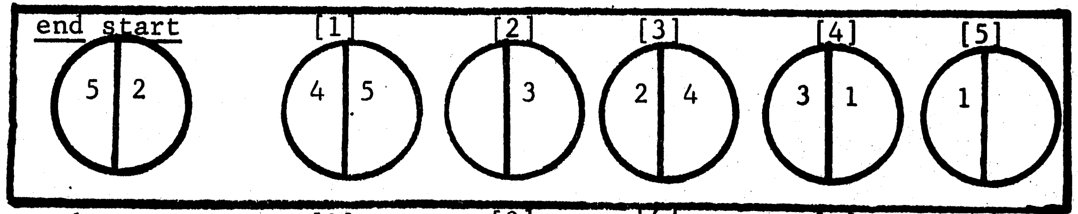
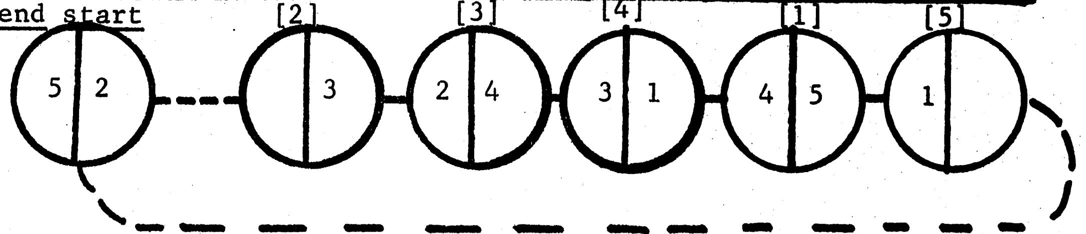


FIGURE III

(logical) b



(physical) a

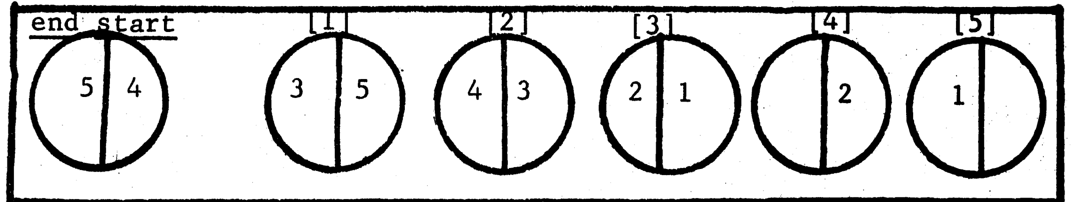
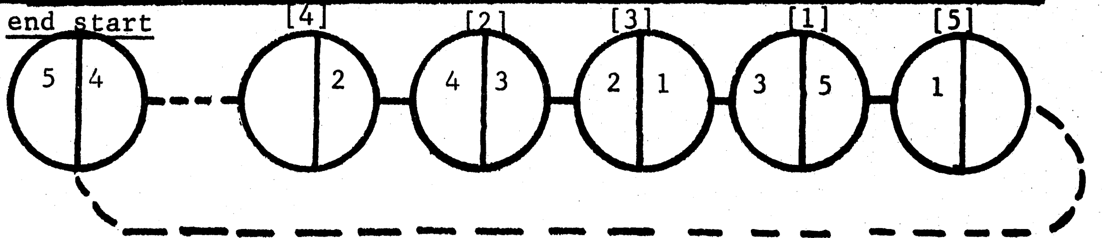


FIGURE IV

(logical) b



(physical) a

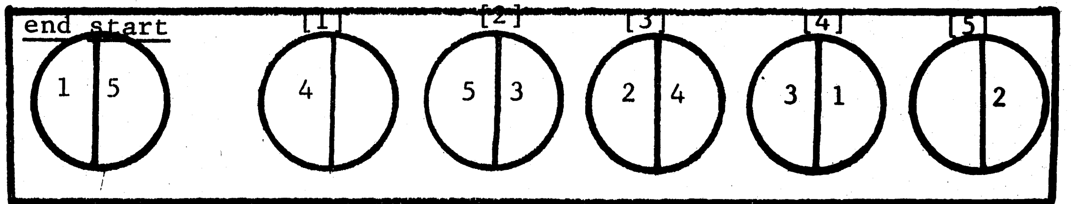


FIGURE V

(logical) b