Published: 9/2/66

Identification

Frames with Complex Structure
J. F. Ossanna, V. A. Vyssotsky, G. G. Ziegler

Purpose

This section describes the general structure of frames and subframes in Multics I/O, and discusses the methods for using structured frames.

Discussion

As has been observed in preceding subsections of section BF.1, each frame attached to a process may be regarded as a sequence of items, and each such frame at any given moment has a current item number. For any frame (e.g. alpha) attached to the process, one may regard the current item number as specifying a frame of data (e.g. beta), consisting of the current item of the containing frame. This arrangement may be declared to the I/O system by

```
call attach('beta','item','alpha')
```

After this attach call, then at any subsequent time the frame beta is the item of alpha which is the current item of alpha at that time. If the current item number of alpha changes, the new current item of alpha is automatically attached as frame beta (and, of course, the old current item of alpha is detached as frame beta). For example, if alpha is a random sectional frame, then following the attach call shown above, the seek call

```
call seek('alpha',3)
```

will attach record 3 of alpha as frame beta. If the next call is

```
call seek('alpha',7)
```

record 7 of alpha will be attached as frame beta, in place of record 3.

Subframes may themselves have subframes. If, for example, beta has been attached as the current item of alpha, the current item of beta may be attached as frame gamma by

```
call attach('gamma','item','beta')
```

Suppose this has been done, and suppose all three frames are random. We shall illustrate the relationship between item numbers in the various frames by an example. To the right of each call in the following sequence is shown the current item numbers in alpha, beta and gamma on return from the call

	cin alpha	cin beta	cin gamma
call seek('alpha',3)	3	1	1
call seek('gamma',6)	3	1	6
call seek('beta',4)	3	4	1
call seek('gamma',6)	3	4	6
call seek('alpha',9)	9	1	1

Any call which changes the current item number in alpha automatically detaches and reattaches beta and gamma, and therefore sets the current item numbers of beta and gamma to 1. Likewise, any call which changes the current item number in beta automatically detaches and reattaches gamma, and thereby sets the current item number of gamma to 1. We shall not pyramid our example further; however, it is worth observing that we could if we wished, attach the current item of gamma as a frame named sam, and so on.

The relationships between the various frames in such a hierarchy, and the restrictions on them, must be well understood. We shall specify these relationships in terms of frames sam and henry, where henry is the current item of sam. We will not say whether sam is the current item of any frame, nor whether the current item of henry is attached as a frame. Our statements hold independent of these considerations. First, if sam is random, henry may be random or sequential. If sam is sequential, henry may be random or sequential. Henry may be attached as a random frame even though sam was constrained to be sequential. More generally, the current item of any frame may be attached as a random frame, even though the containing frame resides on a sequential medium. Next, henry has the trait readable if and only if sam has the trait readable; henry has the trait writeable if and only if sam has the trait writeable. Henry may be declared repositionable whether or not sam is repositionable; however, if sam is repositionable then henry is also repositionable.

If sam is linear, henry is linear. In this case, the maximum size of henry is the element size of sam. The element size of henry need not have any particular relationship to the element size of sam. However, if the element size of sam is not a multiple of the element size of henry, the last element of henry is incomplete; it will be transmitted as an incomplete element on reading or writing. If sam is sectional, it is possible for henry to be either sectional or linear. However, a frame is either sectional or linear, depending on the data in the frame. Hence, if the current record of sam was written as a linear frame or by a write call for sam, then henry is linear. If the current record of sam was written as a sectional frame, then henry is a sectional frame. These statements hold even if the number of bits of data in the current record of sam is zero. If a read call for henry is to succeed, henry must be attached as a linear frame if the data in the current record of sam is linear; henry must be attached as a sectional frame if the data in the current record of sam is sectional. If henry is attached as a linear frame and the data in the current record of sam is sectional, or vice versa, a read or find call for henry will be treated as if henry had never been written. A write or delete call for henry in this situation will cause deletion of the current record of sam (and, if sam is in truncation mode, all following records of sam) followed by writing of henry as a linear or sectional frame according to the way henry is attached. In short, if the attachment of henry does not agree with the data in henry about whether henry is linear or sectional, I/O calls act as if henry contains no data, and execution of a write or delete call destroys any data there was in henry.

The truncation and replacement modes for subframes work as follows. If sam is in truncation mode, a write or delete call for henry causes truncation of sam even if henry is in insertion mode. If sam is in replacement mode, a write or delete call for henry acts as a replacement in sam even if henry is in truncation mode. Within henry itself, write and delete calls cause truncation of henry or replacement into henry exactly as if henry were an independent frame, except in one case. If sam is a linear frame (and so henry is also a linear frame), and if sam is in replacement mode and henry is in truncation mode, and if henry is not the last element of sam, then a write or delete for henry which would normally cause truncation of henry will instead cause the elements of henry which would normally be discarded to be replaced by binary zeros. This case arises because a truncation of henry would imply a truncation of sam, an error since sam was stated to be in <u>replacement</u> mode and a <u>linear</u> frame. Similarly, if henry is the last element of sam, and is shorter than sam, writing or deleting of any element of sam beyond henry will cause henry to be extended by enough binary zeros to make its length the same as the element size of sam.

<u>An Example</u>

Let us consider as an example the problem of programming a PL/I procedure to make a copy of an arbitrary data frame kept in the file system. The procedure is to be called by:

```
call copy(oldfile,newfile,state)
```

where oldfile and newfile are character strings; oldfile is the name of a readable file in the file system, and newfile is the name to be given to the copy to be created. State is a bit string of length 72. In our example we shall simplify error handling by passing the buck up to the caller if any difficulty arises in copying the file. Code to perform the copy operation is shown in figure 1.

The rationale behind the code is as follows. Since the frame to be copied has some structure which is <u>a priori</u> unknown, we must determine its structure as a part of the copying process. We do this by attaching the frame without specifying whether it is to be attached as a linear frame or a sectional frame, and then looking at the status return to find out whether the frame is linear or sectional. If the frame is linear, we attach an output frame as a linear frame, and copy the data. If, however, the frame is sectional, we attach the output frame as a sectional frame. Then we iterate through the records of the input frame, treating each one as a frame of unknown structure in exactly the same way we did the input frame on the level above. This technique will cause recursion to as many levels as there are levels of structure in the data to be copied.

<u>Figure 1</u>

```
copy:     procedure(oldfile,newfile,status);
          declare(oldfile,newfile)character(*);
          declare status bit (72);
          declare (j,k,l,m,many) fixed binary (35);
          declare place pointer;
          declare transit_area bit (2304);
          place=addr(transit_area);
          call copyx(oldfile,newfile,'file',1,status);
          return;

copyx:    procedure(old,new,type,lw,state);
          declare(old,new,type)character(*);
          declare state bit (72);
          declare(to,tn)character(10)varying;
```

<u>Figure 1</u> (Cont.)

```
          to='to'  lw;
          tn='tn'  lw;
          call attach(to,type,old,'r',state);
          if substr(state,1,1) then return;
          call bounds(to,1,0);
          call sizes(to,j,k,l,m);
          call bounds(to,0,k,l,m);
          if substr(state,62,1) then go to sect;
          call attach(tn,type,new,'wl',state);
          if substr(state,1,1)then go to scat;
          call bounds(tn,l,k,0,0,state);
          if substr(state,1,1)then go to leave;
          call seek(to,0,state);
          if substr(state,57,1) then go to leave;
loop:     call read(to,0,place,2304,many,state);
          call write(tn,0,place,many);
          if¬substr(state,49,1) then go to loop;
          go to leave;

sect:     call attach(tn,type,new,'ws',state);
          if substr(state,1,1) then go to scat;
          call bounds(tn,l,k,l,m,state);
          if substr(state,1,1) then go to leave;
          call find(to,0,state);
loopa:    if substr(state,57,1) then go to next;
          lwa=lw+1;
          call copyx(to,tn,'item',leva,state);
          if substr(state,1,1) then go to leave;
next:     call find(tn,1);
          call find(to,1,state);
          if¬substr(state,49,1) then go to loopa;

leave:    call detach(tn,new);
scat:     call detach(to,old);
          return;
          end copy;
```

<u>Another Example</u>

In the Kernel Magnetic Co. reorganizations and personnel transfers are much more frequent than all other events (such as raises) which require changes in personnel records. As a result the company keeps its personnel records as three distinct random data frames in Multics. The basic frame is always attached to the personnel process with the name 'people'. It contains one record for each employee, and the number of that record is the employee's man number. This contains all the personnel data relevant to the employee, except that it contains no information about his position in the organization. A second frame, always attached with the name 'nodes', contains one record for each node on the organization chart. This record consists of three subrecords. The first subrecord contains the man number of the employee at this node in the chart. The second subrecord itself consists of a sequence of subrecords, and each of these contains the number of a node immediately above this one on the chart. (Typically, there is one node above a given node, but there may be zero - there is no node above that occupied by the chairman of the board; or there may be more than one- the node representing the comptroller is immediately below the president, and also immediately below the chairman of the finance committee). The third subrecord of the 'nodes' frame is a sequence of subrecords each of which contains the number of a node immediately below this one on the organization chart. The third frame of personnel data, always attached with the name 'jobs', has one record for each employee, whose record number is the man number of the employee. This record consists of a sequence of subrecords, each containing the node number of one node on the organization chart which the given employee occupies. Typically, the employee holds only one position on the chart, but he may hold more than one. For example, Joe Smith, the shop superintendent of the compressor plant is also the plant manager (and therefore reports to himself as well as to the Vice President, Manufacturing Operations). Joe is also Vice President, Operations, of the semiconsolidated subsidiary Nucleomagnetics, which leases part of the compressor plant; in this capacity Joe reports to the President of Nucleomagnetics.

Now Joe Smith resigns unexpectedly. Company policy states that the organization chart, in this case, shall be modified to make Joe's immediate superior hold Joe's job until a replacement is named. Thus, the Vice President, Manufacturing Operations, now becomes also the plant manager and shop superintendent of the compressor plant, and the President of Nucleomagnetics becomes also the Vice President, Operations of Nucleomagnetics. Since Kernel Magnetics Co. is an old, established organization, such complex operations on the organization chart happen frequently. In fact, it has been necessary to make explicit provision in company policy for the resignation of a man all of whose superiors in one or more of his capacities are himself. The resignation of such a man from those capacities may not be accepted until a replacement is appointed.

Figure 2 shows the PL/I procedure used by the personnel department to cope with the resignation of an arbitrary employee. The operation of the program is basically simple. It has a random frame 'part' attached as the current item of 'nodes', and a random frame 'above' attached as the current item of 'part'. 'Above' is also attached with the stream name 'below' to make the name in the source program correspond to the nature of the data being massaged. The procedure first reads the node numbers of the resigning employee's jobs from 'jobs', and then considers them one at a time. For each job it looks at the list of immediately superior jobs contained in the 'nodes' entry for the job, and tried to find an immediate superior other than the resigning employee himself. If it succeeds, it replaces the man number of the resigning employee, in this particular job, with the man number of the appropriate superior, reduces the number of jobs still held by the employee by one, and goes on to consider his next position. If, however, the resigning employee had no superior other than himself in one of his positions, the procedure goes immediately to consider the next position he holds. When it has so considered all of his positions, the procedure asks two questions. First does the employee still hold any positions? If not, it deletes his entry from the 'jobs' and 'people' frames. Second, if he still holds positions, was he successfully released from any positions during the scan? If so, the scan is done again, since the previous scan may have provided him with an immediate superior other than himself in some position where he had been his only immediate superior. If, however, a scan does not release him from any jobs, then his resignation cannot be accepted, and his records in 'jobs' and 'people' are retained.

Figure 2

```
resign:   procedure(nemo);
          declare nemo fixed binary (35);
          declare (ta(1000),tb(1000)) fixed binary (35);
          declare state bit (72) static;
          declare point pointer;
          call seek('jobs',nemo);
          call shift('hisjobs',ta,ma);
          call delete('jobs',nemo);
          na=ma;
wot:      ka=0;
how:      do ia=1 to ma;
          if ta(ia)=0 then go to bot;
          call seek('nodes',ta(ia));
          call seek('part',2);
          call shift('above',tb,mb);
may:      do ib=1 to mb;
          call seek('nodes',tb(ib));
          call read('part',1,man,1);
          if man¬=nemo then go to fit;
          end may;
          go to bot;
```

Figure 2 (Cont.)

```
fit:      call seek('nodes',ta(ia));
          call write('part',1,man,1);
          ta(ia)=0;
          ka=ka+1;
bot:      end how;
          na=na-ka;
          if na=0 then go to delete;
          if ka=¬0 then go to wot;
retain:   ja=1
gap:      do ia=1 to ma;
          if ta(ia)=0 then go to pig;
          point=addr(ta(ia));
          call write('hisjobs',ja,point,1);
          ja=ja+1;
pig:      end gap;
          return;
delete:   call delete('people',nemo);
          return;
shift:    procedure(from,to,many);
          declare from character(*);
          declare to (*) fixed binary (35);
          ngot=0;
          many=0;
          nmany=0;
zap;      many=many+ngot;
          nmany=nmany+1;
          point=addr(to(many+1));
          call read(from,nmany,point,1,ngot,state);
          if substr(state,49,1)='0'b then go to zap;
          return;
          end shift;
          end resign;
```