Published: 03/14/69

Identification

I/O Outer Calls (BF.1.00)
P.G. Neumann, M.A. Padlipsky

Note that the following calls are entry points to the I/O Switch (<iosw>). They are callable through the gate segment <ios> (BS.0.03).

See BS.5.00 for declarations of the generic I/O arguments. The definitions of the arguments are contained in the functional descriptions below.

$attach

The attach call associates the given ioname (ioname1) with a previously defined name or otherwise known device specified by ioname2. This association is meaningful within the framework of the user's process group. The resulting attachment remains in force until removed by a detach call (see below). A type and a mode (see the changemode call below) are associated with the attachment. See BF.1.01.

```
call attach(ioname1,type,ioname2,mode,status);
```

$detach

The detach call removes for the given ioname(s) an association established by an attach call. The disposal argument indicates how dedicated resources (e.g., tapes and tape drives) are to be treated. See BF.1.01.

```
call detach(ioname1,ioname2,disposal,status);
```

$changemode

The mode (specified by mode) of an attachment describes certain characteristics related to the attachment (e.g., readable; writable; appendable; random or sequential; if logical, linear or sectional). The changemode call permits mode changes to be invoked for the given ioname(s) which modify the mode of the attachment. See BF.1.01.

```
call changemode(ioname,mode,status);
```

$getmode

The getmode call returns a terse encoding (bmode) of the mode of the attachment specified by the given ioname. This call is intended primarily for use by IOS modules. See BF.1.01. (Design is tentative.)

```
call getmode(ioname,bmode,status);
```

$readsync

For a given valid ioname (i.e., a name which has previously been properly attached by means of an attach call), the readsync call sets the read synchronization mode (rsmode) of subsequent read calls (see below). This mode is either synchronous or asynchronous. Synchrony implies that control is not returned to the caller until the read request is either physically initiated or physically completed, depending upon whether the workspace synchronization mode (see the worksync call below) is asynchronous or synchronous, respectively. Asynchrony implies that read-ahead is possible to the extent permitted by the limit argument, which points to the desired maximum number of elements which may be read ahead. The default mode is asynchronous. See BF.1.04.

```
call readsync(ioname,rsmode,limit,status);
```

$writesync

For a given (valid) ioname, the writesync call sets the write synchronization mode (wsmode) of subsequent write calls (see below). The mode is either synchronous or asynchronous. Synchrony implies that control is not returned to the caller until the write request is either physically initiated or physically completed, depending upon whether the workspace synchronization mode (see worksync) is asynchronous or synchronous, respectively. Asynchrony implies that write-behind is possible to the extent permitted by the limit argument, which points to the desired maximum number of elements which may be written behind. The default mode is asynchronous. See BF.1.04.

```
call writesync(ioname,wsmode,limit,status);
```

$resetread

The resetread call is used to delete unused read-ahead data collected by the I/O system as a result of read-ahead associated with the given ioname. See BF.1.04.

```
call resetread(ioname,status);
```

$resetwrite

The resetwrite call is used to delete unused write-behind data collected by the I/O system as a result of write-behind associated with the given ioname. See BF.1.04.

```
call resetwrite(ioname,status);
```

$worksync

For a given ioname, the worksync call sets the workspace synchronization mode. The mode (wkmode) is either synchronous or asynchronous. Synchrony implies that control is not returned to the caller until the I/O system no longer requires the user's workspace (see read and write calls below). Asynchrony implies that some kind of initiation of the call has taken place, although the workspace may still be in use. The default mode is synchronous. See BF.1.04.

```
call worksync(ioname,wkmode,status);
```

$iowait

For a given ioname whose workspace synchronization mode is asynchronous, the iowait call defers the return of control as if the workspace synchronization mode were synchronous for the most recent read or write call or for a specified previous call. The argument oldstatus is the original status argument returned for the particular previous transaction, and is used to identify that transaction uniquely. If oldstatus is missing, the most recent transaction is implied. See BF.1.04.

```
call iowait(ioname,oldstatus,status);
```

$abort

When the workspace synchronization mode is synchronous, the abort call causes all outstanding transactions to be aborted (oldstatus is ignored). When the workspace synchronization mode is asynchronous, transactions are aborted beginning with the one corresponding to oldstatus, which contains the identification of an earlier call. See BF.1.04.

```
call abort(ioname,oldstatus,status);
```

$order

The order call is used to issue a request (request) to outer modules. argptr points to a data structure containing arguments relevant to the particular request. The call is used for communication among I/O system modules. It may also be used to set hardware device modes.

```
call order(ioname,request,argptr,status);
```

$getsize

The getsize call returns the current element size (elsize) associated with read and write calls for the given ioname. See BF.1.05.

```
call getsize(ioname,elsize,status);
```

$setsize

The setsize call sets the element size (elsize) for subsequent read and write calls with the given ioname. See BF.1.05.

```
call setsize(ioname,elsize,status);
```

$read

The read call attempts to read into the specified workspace (starting offset items from the beginning of the workspace) the requested number (nelem) of elements from the frame specified by the given ioname. Reading begins with current item of frame. Thus for a linear frame, reading begins with the element pointed to by the "read" pointer. Reading is normally terminated by the occurrence of a read delimiter or by the reading of nelem elements, whichever comes first. The "read" pointer is moved to correspond to the element after the one last read. For a sectional frame Y, reading begins with the first element (pointed to by the "read" pointer for X) of the current subframe X, where the current subframe is that pointed to by the "current" pointer for the frame Y of which X is a subframe. Reading is normally terminated by the occurrence of the end of the subframe, by the occurrence of a read delimiter, or by the reading of nelem elements, whichever comes first. The "current" pointer for Y and the "read" pointer for X are moved to correspond to the first element of the next frame X. See BF.1.06.

```
call read(ioname,workspace,offset,nelem,nelemt,status);
```

$write

The write call attempts to write from the specified workpace (starting offset items from the beginning of the workspace) the requested number (nelem) of elements onto the frame specified by the given ioname. The number of elements actually written is returned (nelemt). The behavior of the write call with respect to the "write" pointer is similar to that described above for the read call with respect to the "read" pointer, except that there is no write delimiter. Writing begins with the current item of the frame. Thus for a linear frame, writing begins with the element pointed to by the "write" pointer. Writing is normally terminated by the writing of nelem elements. The "write"pointer is moved to correspond to the element after the last one written. For a sectional frame Y, writing begins with the first element (pointed to by the "write" pointer for X) of the current subframe X, where the current subframe is that pointed to by the "current" pointer for the frame Y of which X is a subframe. Writing is normally terminated by the writing of nelem elements. The "current" pointer for Y and the "write" pointer for X are moved to correspond to the first element of the next frame X. See BF.1.06.

```
call write(ioname,workspace,offset,nelem,nelemt,status);
```

$setdelim

The setdelim call establishes elements which delimit data read by subsequent linear read calls with the given ioname. The argument breaklist points to a list of break characters (containing nbreaks elements), each serving simultaneously as an interrupt, canonicalization and erase-kill delimiters. Break characters are meaningful only on character-oriented devices. The argument readlist points to a list of read delimiters (containing nreads elements). The new delimiters established by this call are in effect until superseded by a subsequent setdelim call. See BF.1.06.

```
call setdelim(ioname,nbreaks,breaklist,nreads,readlist,
              status);
```

$getdelim

The getdelim call returns to the caller the delimiters established by the most recent setdelim call, with the arguments having precisely the same meaning for both calls. See BF.1.06.

```
call getdelim(ioname,nbreaks,breaklist,nreads,readlist,
              status);
```

$seek

The seek call sets the reference pointer specified by ptrname1 to the value of the pointer specified by ptrname2 plus the value of a signed offset (if offset is present). ptrname1 may be "read", "write", "last" or "bound", or in the case of a sectional frame, "current", "last" or "bound". ptrname2 may be "read", "write", "first", "last" or "bound", or in the case of a sectioal frame, "current", "first", "last" or "bound". For physical I/O (using the readrec and writerec calls), ptrname1 may be "currentrec" "last" or "bound", while ptrname2 may be "currentrec", "first", "last" or "bound". The seek call is used to truncate, e.g., seek(ioname,"last","last",-40), or to set the bound of the frame, e.g., seek(ioname,"bound", "last",27), in addition to its more traditional usage involving the "read" and "write" pointers, e.g., seek (ioname,"read","write",-2). The "read" and "write" pointers are also set as a result of read and write calls, respectively (see above). Each reference pointer refers to an item number. Which frame is referred to depends upon the type argument of the attach call. See BF.1.06.

```
call seek(ioname,ptrname1,ptrname2,offset,status);
```

$tell

The tell call returns the value of the pointer specified by ptrname1 as an offset (offset) with respect to the given ptrname2. The arguments ptrname1, ptrname2 and offset have the same meaning as in the seek call. As an example, the tell call may be used to obtain the bound of a frame by call tell(ioname,"bound","first",offset). See BF.1.06.

```
call tell(ioname,ptrname1,ptrname2,offset,status);
```