CONTENTS

Wednesday, April 8, 1987

3.3 Intrinsic Interface Definitions

3.3.1 IPCCreate

Function:

Creates a socket.

Declaration:

```
PROCEDURE IPCCreate (      socket_kind : integer;
                           protocol    : integer;
                       var flags       : flags_type;
                       var opt         : opt_type;
                       var sd          : descriptor_type;
                       var result      : integer );
```

socket_kind (input)

Defines the type of socket to be created. Valid types are:

3 - call
4 - pxp request (privileged users only)
5 - pxp reply   (privileged users only)

Datagrams are not supported.

protocol (input)

Defines the protocol module the user will be interfacing to. If this value is zero, a default protocol will be used. For call sockets, the default protocol is TCP. Supported values are:

2 - X.25
4 - TCP
6 - HP-PXP
13 - OSI'SS

For OSI'SS, TCP and X25 the only allowed socket type is call. Any others will produce a sockerr 10

flags (input)

32 bits, each specifying an optional capability. The only flag which is currently supported is:

flags [protect] (bit 0, input)

If this bit is set, then the socket will be privileged

opt (input parameter)

Array of options, constructed with ADDOPT. The options currently allowed are:

max_msg_size (code=1, len= 2, 2 byte integer) (input)

> This option defines the maximum expected message size. This opt can only be used with datagram, pxp request, or pxp reply sockets This option does not necessarily limit the user to the specified size. it is intended to be used by IPC and/or the protocol module in determining how much resources should be allocated for a user. [illegible]ven protocol may or may not take advantage of this information

max_conn_reqs_queue [illegible] len=2, 2 byte integer (input)

> Used to specify the maximum number of unreceived connection requests that [illegible] be queued to a call socket. Valid only for connection oriented protocols.

max_mesgs_queue_in (code=7, len=2, 2 byte integer) (input)

> For datagram or pxp sockets. This parameter specifies the number of unreceived messages the user expects to be queued to this socket at any one time. This does not necessarily limit the user to this number; it is intended to be used by IPC and/or the protocol module in determining what resources need to be allocated for the user.

pxp_retry_count code=9, len=2, 2 byte integer) (input)

> Specifies the number of retries for a pxp request socket.

pxp_timeout_val (code=10, len=2, 2 byte array) (input)

> Specifies the timeout value for pxp request sockets.

protocol_rel_addr (code=128, n-byte integer) (input)

> The protocol relative address assigned to the newly created socket. For privileged users, the address may be up to 16 bytes long. For non-privileged users, the address must be exactly 2 bytes (otherwise sockerr 165), and the address must be in the range %74057 to %77777 (otherwise sockerr 164). This option tells IPC to create the socket with the specified protocol address rather than dynamically allocating an address.

X25_net_name (code=140, len=8, 8 character array) (input)

> Specifies the X25 network interface name, telling IPC which X25 instance the socket should be identified with. For X25 call sockets, this option is required. Otherwise, a sockerr 141 will result. IPC automatically upshifts the characters in the supplied name to all capital letters.

protocol_flags (code=144, len=4, 4 byte integer) (input)

32 bits of protocol-specific flags. Currently, only X25 uses these flags. For IPCCREATE, only one is defined; any others will cause a SOCKERR 155.

catch-all-socket (bit #2) (input): Defines the socket as a catch-all. The catch_all_socket is a CALL socket that gets all incoming calls whose relative address is not assigned to a specific socket. Only one catch-all socket can be defined for each directly connected X25 network. This flag is meaningless for permanent virtual circuits and also for the process which is initiating a call. (privileged users only)

sd (output)

Socket descriptor. Value returned which is to identify the created socket

result (output)

Resultant error code, else zero.

Discussion:

The IPCCREATE intrinsic is called to create a socket. The intrinsic returns a socket descriptor which is used to identify the socket when using other IPC intrinsics.

The relative address field is used by X25 to find the CALL socket an incoming call is intended for. X25 uses the first four bytes of the connection call user data (CUD) field as an address. This address is matched to the relative address asssigned to all the X25 call sockets If a call socket's relative address matches the CUD address then the incoming call is routed to that socket. If no match is found then the incoming call is routed to the catch all socket. The catch all socket is defined by creating a socket with the catch all flag set. Only one catch all socket can be defined per directly connected network. Finally, if the CUD address does not match any call sockets and no catch all socket is defined then the incoming call is cleared.

If a non-privileged user wants to specify the address to be assigned [illegible] new socket, the range %74057 to %77777 must be used. For a privileged [illegible], there is no specific re[illegible]on on the address range. However, it is recommended that only addresses between %1 and %74056 be used. This will prevent duplication of addresses with those that are automatically allocated when the user does not provide opt 128.

3.3.2 IPCNAME

Function:

Associates a name with a socket.

Declaration:

```
PROCEDURE IPCNAME (     sd          : descriptor_type;
                    VAR socket_name : socket_name_type;
                        nlen        : integer;
                    VAR result      : integer );
```

Parameters:

sd (input)

Socket descriptor of socket to be named. If a connection descriptor is given, SOCKERR #30 will result; for a pxp request descriptor, a SOCKERR #104 will occur

socket_name (input [illegible]put)

Name to be given to socket.

nlen (input)

Byte length of specified socket name. Maximum is 16.

If the specified name length is zero, an eight byte name will be randomly generated for the user and returned in the socket_name parameter.

result (output)

Resultant error code, else zero.

Discussion:

The IPCNAME intrinsic allows a user to bind a name to a socket. Using the IPCLookUp intrinsic, another user can obtain access to the socket by knowing its name. The syntax of the socket name is defined in Section 3.2.1.

Note that this intrinsic binds a name to a socket, not an address. That is, when the socket is destroyed the name will be removed from the registry.

Up to four names may be assigned to a single socket. If a fifth name is attempted, SOCKERR #35 will occur. However, the same name may not be used for multiple sockets. If IPCNAME is called with a name which already is used, a SOCKERR #31 will result.

This intrinsic may not be called in split stack. Otherwise, SOCKERR #32.

All alpha characters in the name are automatically upshifted. Therefore, names may not be distinguished by case of the alpha characters.

If the intrinsic call fails for any reason, the condition code will be set to CCL.

3.3.3 IPCNAMERASE

Function:

To delete a socket name.

```
PROCEDURE IPCNAMERASE ( VAR socket_name : socket_name_type;
                            nlen        : integer;
                        VAR result      : integer );
```

Parameters:

socket_name (input)

Name currently bound to a socket that is to be removed.

nlen (input)

Byte length of specified socket name. Maximum is 16.

result (output)

Resultant error code, else zero.

Discussion:

If a socket has been named with the IPCNAME intrinsic, the owner of the socket may remove the name with the IPCNAMERASE intrinsic. If the user is not the owner of the socket which has the specified name bound to it, the intrinsic will terminate with a SOCKERR #38. If the name is not found (no socket has the name), a SOCKERR #37 will occur.

If the intrinsic fails for any reason, the condition code will be set to CCL.

3.3.4 IPCLookUp

Function:

To obtain a destination descriptor for a named call socket.

Declaration:

```
PROCEDURE IPCLookUp ( VAR socket_name     : socket_name_type;
                          nlen            : integer;
                      VAR location        : location_type;
                          loclen          : integer;
                      VAR flags           : flags_type;
                      VAR dest_descriptor : descriptor_type;
                      VAR protocol        : integer;
                      VAR socket_kind     : integer;
                      VAR result          : integer );
```

Parameters:

socket_name (input)

Name of the socket for which the search will be conducted and for which a destination descriptor will be defined.

nlen (input)

Byte length of the supplied socket name. Maximum length is 16. If the nlen is not at least one or is greater than 16, sockerr #28 will be returned.

location (input)

If specified, this is the name of the node where the socket is assumed to reside. This is the node on which a search for the specified name will occur. If not given, then the search will take place locally. This parameter is optional. However if it is given then the loclen must also be given. Otherwise a sockerr #27 will occur

loclen (input)

Byte length of the node name. This parameter is optional, but if it is given then the location must also be given. The loclen parameter may be zero indicating the lookup is to take place on the local node. The maximum length is 50. Otherwise, a sockerr #39 will occur.

flags (input)

Various option flags. Only one is defined:

flags [protected] (bit #0, input)

If this flag is set, the destination descriptor will be protected and can only be accessed by privileged users. If this flag is set by a non-privileged user, sockerr #7 will result.

dest_descriptor (output)

Descriptor which can be used by the calling process to access the socket that had been looked up. This descriptor is required by other intrinsics, including IPCCONNECT

protocol (output)

Protocol id resulting from the socket search. Identifies the level 4 protocol module used by the socket.

socket_kind (output)

Socket type which was found [illegible] search.

result (output)

Resultant error code, else zero.

Discussion:

The IPCLOOKUP intrinsic is used to gain access to a socket whose name is known. (The name was previously defined with an IPCNAME operation.) The destination descriptor is associated with that name, and the calling process thereafter uses the descriptor to access the named socket.

In addition to the socket's name, the user may specify where the registry search is to take place. If no location is specified the registry search will be performed on the local node If the name is not found, a sockerr #37 will be returned.

For a remote search, the protocol and socket type combination must be valid. Currently, the only combinations which will not result in a sockerr #46 are call socket with TCP protocol, or pxp reply socket with pxp protocol.

Required parameters are socket_name, nlen, and dest_descriptor.

3.3.8 IPCRECVCN

Function:

To receive a connection request on a call socket.

Declaration:

```
PROCEDURE IPCRecvCn (     sd                : descriptor_type;
                      VAR cd                : descriptor_type;
                      VAR flags             : flags_type;
                      VAR opt               : opt_type;
                      VAR result            : integer );
```

Parameters:

sd (input)

Socket descriptor for a call socket.

cd (output)

Connection descriptor identifying the local endpoint of the connection which is established with the call to this intrinsic.

flags (input/output)

Option flags. 32 bits. Defined are:

flags [protect] (bit 0, input)

If this bit is set, the created connection descriptor will be protected and can only be accessed by privileged users.

flags [tcpmsg] (bit 1, input)

If true, then TCP is instructed to operate in message mode. otherwise, stream mode will be used on the connection.

flags [no_output_flags] (bit 16, input)

If true and this intrinsic is call in nowait mode, the flags parameter will not be updated when the intrinsic completes.

flags [defer] (bit 18, input)

If set, completion of the connection will be deferred, and the user can later decide whether to accept or reject the connection with IPCCONTROL.

flags [checksum] (bit 21, input)

If set then the protocol module will be instructed to enable checksumming on the established connection. (Currently, the only protocol module which supports checksumming is TCP.)

flags [discarded] (bit 25, output only)

This flag indicates that call user data was present, but some or all of it had to be discarded. This occurs when no call_user_data_recv option was given or if the space [illegible]ded was to small to hold the data received If flags bit #16 is set on the initial call to IPCRECVCN and nowait mode [illegible] used, the discarded flag will not be output when the intrinsic completes.

flags [vectored] (bit [illegible], input)

If set, then the received call user data is expected to be vectored, meaning that the data will be placed in a memory location specified by the user rather than directly into the buffer of the option entry. The target location is specified with one or two vectors which indicate a DST and offset.

opt (input/output parameter)

Byte array containing various options. The entries are assembled with an INITOPT/ADDOPT sequence. The defined opts are:

max_send_size (code=3, len= 2, 2 byte integer) (input)

This option may be used to inform the protocol of the length of the largest message to be sent by the user on this connection. The default is 1024 for TCP.

max_recv_size (code=4, len=2, 2 byte integer) (input)

This option may be used to inform the protocol of the length of the largest message to be received by the user on this connection. The default is 1024 ro TCP.

call_user_data_recv (code=5, len=n, n byte buffer) (output)

This option specifies that call user data may be received during the connection establishment. (Not supported by TCP.) The data can be either vectored or non-vectored, depending upon the state of flags bit 31 when the intrinsic is called. If not vectored, the data will be returned into this buffer area of the opt array. See IPCCONNECT for a descripton of the vector format. The maximum non-vectored length is 512 bytes.

The actual byte count received is available to the user. If the data is not vectored, then the byte count will be placed in the length parameter of the option entry and may be determined with a READOPT intrinsic call. If the data is vectored, then the location of the received count depends upon whether nowait I/O is being used. If waited, then the count is put in the length parameter of the FIRST vector (which is assumed to still be in its original place in the opt entry). This will be the number of TOTAL bytes received, even if there were two vectors. If nowait I/O is selected and the data is vectored, then the total byte count will be available in the tcount parameter of the IOWAIT.

If not enough buffer space was allocated for the actual amount of call user data received, then the discarded flag will be set.

send_burst_size (code=134, len=2, 2-byte integer) (input)

Informs the protocol module of the send burst size to be used. The integer must be in the range 1 to 7, setting the number of messages which can be sent on this connection to the remote node without the remote peer actually accepting them. The default send burst size is 3 for TCP.

recv_burst_size (code=135, len=2, 2-byte integer) (input)

Informs the protocol module of the receive burst size. The integer must be in the range 1 to 7, with a default of 3 for TCP. The local protocol module is instructed to accept up to this number of incoming messages, even if the local receiver has not yet processed them.

update_threshold (code=136, len=2, 2-byte integer) (input)

The integer value is sent to the protocol module to specify how the receive window is to be updated. The integer value must be in the range of 0 to 100. This specifies a percentage of the total window size which must be available before an update packet is sent to the remote node. (TCP only at this time.)

calling_node_addr (code=141, max len=8, 8 byte buffer) (output)

If this opt is specified, the protocol module is requested to supply the address of the calling node. This is primarily an X25 [illegible] e, although IPC does no protocol checking on this opt. If the [illegible] th is no 8, then SOCKERR 144 will occur.

protocol_flags (code=144, len=4, 4 byte buffer) (input/output)

This opt contains 32 bits of protocol-specific flags. If the length is not 4 bytes, then SOCKERR 155 is returned. Currently, only X25 uses these flags, although IPC does no protocol checking on this opt. If flags bit #16 is set on the initial call to IPCRECVCN and nowait mode is being used, the protocol flags will not be output when the intrinsic completes. The defined flags are:

pad_call (bit #14) (output):

If this bit is set, the the protocol module has received an indication that the connection request originated from a pad. The

X25 protcol module makes this determination by examining bit #1 of the first octet in the call user data field.

calling_node_add_available (bit #16) (output):

This flag indicates that the calling node address was present in the call request message. The address will be placed in the appropriate entry of the opt array.

q_bit_flag (bit #19) (output):

Indicates the state of the Q bit in the X25 packet received by the protocol module. For connection establishment with X25, this bit should never be set.

result (output)

Indicates whether the request was successful. If nowait I/O is used, the result parameter will give information about the initial call to the intrinsic, but the parameter will not be updated upon final completion of the request. To examine the result of the IPCRECVCN completion, the user can call IPCCHECK to view both the protocol module error and the IPC error.

Discussion:

The IPCRECVCN intrinsic allows users to receive connection requests and establish a VC socket (identified by the returned connection descriptor). The user can then use the IPCSEND and IPCRECV intrinsics to send and receive data on the connection.

The call user data which can be received on connection establishment can be either vectored or not. There are two primary reasons for using vectored data: 1. The user wants the data to be scattered into two different locations, perhaps because the meaning of the two portions is different, 2. The user does not want the data to be returned into the stack area. This can be important if the available stack-relative space is limited or if the user cannot guarantee that upon completion of a nowait call to this intrinsic that the location of the original opt array will still be valid.

3.3.7 IPCConnect

Function:

Initiates a connection request.

Declaration:

```
PROCEDURE IPCConnect (     sd              : descriptor_type;
                           dest_descriptor : descriptor_type;
                       VAR flags           : flags_type;
                       VAR opt             : type_opt;
                       VAR cd              : descriptor_type;
                       VAR result          : integer );
```

Parameters:

sd (input)

Socket descriptor. Refers to a call socket the user has previously created. This parameter is optional. If it is not given or if it is -1, then a "ghost" call socket will be created by IPC for the purpose of establishing the connection. This is a temporary socket which will be closed when IPCCONNECT completes. If the destination node is using X25 protocol, then a call socket must be specified (no ghost socket is allowed). Otherwise, a SOCKERR 27 will be given.

dest_descriptor (input)

Descriptor which the user has previously obtained that identifies the socket that is to receive the connection request.

flags (input)

32 bits, specifying various optional capabilities. Supported flags are:

flags [protect] (bit 0, input)

If this bit is set, then the connection will be protected and can only be accessed by privileged users.

flags [tcpmsg] (bit 1, input)

If set, this bit tells the TCP protocol to operate in message mode. If set false, then TCP will be in stream mode (default). This capability is available only to privileged users (SOCKERR 7 for non-privileged users).

flags [checksum] (bit 21, input)

If set, the protocol module is instructed to use checksumming on the connection. Currently, only TCP supports this feature. Note that checksumming will degrade performance of the data transfers.

flags [vectored] (bit 31, input)

If set, this flag specifies that the call user data in the opt array will be vectored. If vectored, then the CUD will be located in user b[illegible]rs (u[illegible]o 2 buffers allowed), and the opt entry will contain [illegible]rs to [illegible]ese buffers

opt (input)

Array of options, defined with an ADDOPT. Entries with a code other than those listed here cause an error. This includes a code of zero.

call user_data_send (code=2, len=n, n byte array)   input

Data to be sent when the connection is establishing. The data may be vectored or not, according to the state of flags [vectored]. If not vectored, the actual data will be in the opt entry. The maximum length of non-vectored data is

X25 protocol with noaddress flag set: 16 bytes

X25 protocol with noaddress flag not set: 12 bytes

all other pr[illegible]ols: 512 bytes

If vectored, the data will reside in user buffers (2 maximum), and the information in the opt entry will be vectors to the buffers. A vector consists of four words and has the following format:

```
 ------------
|    TYPE    |   Type:
|------------|          0 - Address is stack relative
|    DST     |          1 - Address is relative to a data
|------------|              segment index as returned b.
|   OFFSET   |              the GETDSEG intrinsic. The
|------------|              user must be privileged.
| BYTE COUNT |            - Address is relative to the
 -----------                specified DST. The user must
                            be privileged.

                 Offset:
                            A DB-relative byte ofset for type 0.
                            For types 1 and 2, an offset relative
                            to the start of the DST.

             Byte Count:
                            The length of the user buffer
```

For vectored data, the length parameter of the opt entry (specified with an ADDOPT) must be either 8 or 16, indicating the length of the vector(s).

max_send_size (code=3, len= 2, 2 byte integer) (input)

This option may be used to inform the protocol of the length of the largest message to be sent by the user on this connection. The integer must be in the range 1 to 32,000. If the value is smaller than previously set, the option will be ignored. If this option is not specified, the protocol module will default to a send message size of 1024.

max_recv_size (code=4, len=2, 2-byte integer) (input)

This option may be used to inform the protocol of the length of the largest message to be received by the user on this connection. The valid range is 1 to 32,000 with a default of 1024. If the value is smaller than previously set, the new value will be ignored.

protocol_rel_address (code=128, len=2, 2-byte integer) (input)

Allows the user to define the source address for the connection. If the user is not privileged, the address must be in the range %74057 to %77777. Otherwise a SOCKERR 164 will be returned. The address length must be two bytes (otherwise SOCKERR 165).

send_burst_size (code=134, len=2, 2-byte integer) (input)

Informs the protocol module of the send burst size to be used. The integer must be in the range 1 to 7, indicating the number of messages which can be sent to the remote node without that node having processed them. Default is 3. (Privileged users only.)

recv_burst_size (code=135, len=2, 2-byte integer) (input)

Informs the protocol module of the receive burst size. The integer must be a number in the range 1 to 7. This sets the number of messages which may be received without the local user having processed them. This burst size is used to calculate the window size which TCP advertises to the remote end. (Privileged users only.)

update_threshold (code=136, len=2, 2-byte integer) (input)

The integer value is sent to the protocol module to specify how the receive window is to be updated. The value must be between 0 and 100, indicating the percentage of the total window size which must be available before TCP will send an update packet to the remote end. The default is 50%.

facilities_set_name

(code=142, len=8, packed array of 8 characters max) (input)

This option allows the user to specify a facilities set name which will be associated with the connect·ion (an X25 capability).

protocol_flags (code=144, len=4, 4 byte buffer) (input)

The bits of this four-byte option are taken as flags which are unique to the protocol. If any undefined flags are specified, SOCKERR 155 will occur. The only currently-defined flag bit is:

no_address (bit #17 of the double word option entry)

If set, this flag allows the maximum length of the X25 call user data to be 16 bytes. Otherwise, the maximum is 12 bytes for X25.

cd (output;

Connection descriptor. Returned value w[illegible] is used in succeeding intrinsics to identify the connection.

result (output)

Resultant error code else zero.

Discussion:

This intrinsic is used to establish a connection. The user will generally have a call socket to use in the intrinsic call. However, if no call socket is specified, then a "ghost" socket will automatically be created and used for the connection initiation. This ghost socket will be destroyed before the IPCCONNECT completes. However, ghost sockets are not allowed for X25.

If the protocol is not TCP or OSI'SS then the address of the call socket will be used for the source. For TCP and OSI'SS, the user may specify the source address. Non-privileged users are limited in the range of the address which can be specified. Privileged users are not limited to a certain range, but it is suggested that they use only addresses between %1 and %77777. This will prevent possible overlap with any addresses which are automatically allocated when the user does not specify the source address. If the TCP/OSI'SS user does not specify the address, then one will be allocated in the range %100000 to %123777. (Addresses %124000 to %177777 are used for connection sockets.)

A successful result only means that the connection request has been initiated. The user must call IPCRecv with cd to determine the success or failure of the request.

Use and specific meaning of the options is determined by the actual protocol implementation.

Burst sizes for user sockets are only supported for protocols which preserve message boundaries. They may also be supported for NS applications using 'message mode' TCP.

To establish a connection, the destination socket must also be a call socket using the same protocol.

3.3.9 IPCSend

Function:

Sends data on a connection.

Declaration:

```
PROCEDURE IPCSend (     cd      : descriptor_type;
                    VAR data    : data_buffer;
                        dlen    : integer;
                    VAR flags   : flags_type
                    VAR opt     : opt_type;
                    VAR result  : integer );
```

Parameters:

cd (input)

Connection descriptor which identifies the virtual circuit to be used for the send. If the connection is shared by more than one process, the first word of cd must contain the process identification number of the process which created the connection. If the current process is the connection creator, then the first word may be zero.

data (input)

This parameter contains either the actual data to be sent or vectors pointing to users buffers which contain the data. If flags bit #31 is set, then the data is assumed to contain one or two vectors.

A vector consists of four words and has the following format:

```
     ------------
    |    TYPE    |   Type:
    |------------|         0 - Address is stack relative
    |    DST     |         1 - Address is relative to a data
    |------------|             segment index as returned by
    |   OFFSET   |             the GETDSEG intrinsic. The
    |------------|             user must be privileged.
    | BYTE COUNT |         2 - Address is relative to the
     ------------              specified DST. The user must
                               be privileged.

                     Offset:
                               A DB-relative byte offset for
                               type 0, otherwise relative to
                               start of DST

                 Byte Count:
                               The length of the user buffer
```

Note that if the data is to be vectored, the 'data' array must contain exactly 8 or 16 bytes, and the 'dlen' paramter must be either 8 or 16.

dlen (input)

If the data is not vectored, then dlen must be greater than one and not greater than %72460. This is the byte count of the data. If the data is vectored, then dlen must be either 8 or 16, indicating the byte length of the vectors contained in the data parameter.

flags (input only)

Option flags. Defined are:

flags [shared_conn] (bit #0, input)

This flag indicates that the connection specified by the cd parameter is being shared by more than one process. In this case, the first word of cd must indicate the PIN of the process which owns the connection. This is a privileged function; if this flag bit is set and the user is not privileged, a sockerror 7 will result.

The only sender on a shared connection who can use nowait I/O is the connection owner. To use nowait I/O on a shared connection, the owner must call IPCSEND without the shared_conn flag set.

flags [more_data] (bit #26, input)

This bit is intended for use with stream protocols to provide the user some control over the buffering and transmission of data at the sender's end of the connection. If set, the protocol module should expect more data to be sent. If this bit is not set, then the protocol module is instructed to send (push) the data immediately. If set, then the protocol module can use its own algorithm to decide how to concatenate and send the user data. The reader is directed to the documents for the protocol module for a complete description of "normal" stream mode. (The initial implementation of TCP will always push, so this bit has no effect with that protocol.)

flags [vectored] (bit #31, input)

If this bit is set then the data to be sent is to be gathered from the addresses given in the data parameter. Up to two user buffers may be specified from which the data will be taken.

opt (input)

Array of options, assembled with an ADDOPT intricsic. Defined are:

data_offset (code=8, len=2, 2 byte integer, input option)

Defines a byte offset from the data parameter's address where IPC is to begin looking for the data. This opt must not be used if the data is to be vectored!

protocol_flags (code=144, len=4, 4 byte buffer) (input only)

This op[illegible]tains 32 bits of protocol-specific flags. Currently, only X25 uses these flags, although IPC does no protocol checking on this opt. The defined flags are:

d_bit_flag (bit [illegible] input): Specifies the state of the D bit in the X25 packet. [illegible]e [illegible]oit is used to request end-to-end acknowledgement of the data.

q_bit_flag (bit #19) (input): Specifies the state of the Q bit in the X25 packet to be sent by the protocol module. The Q bit is used to mark the data as control information intended for a pad.

urgent_data_flag (bit #27) (input): If set, this bit will cause the data to be marked as urgent

Any flag[illegible]ther than these will cause a SOCKERR 155.

Discussion:

This intrinsic may be called in split stack.

Up to seven output operations may be pending at one time per connection.

If the connection is being shared, IPCSEND will always be a blocking operation; nowait will not be in effect.

Note that a connection may be shared only for sends. The receiving end of a connection may not be shared.

3.3.10 IPCRECV

Function:

To receive a reply to a connection request or to receive data on an established connection.

Declaration:

```
PROCEDURE IPCRecv (     cd     : descriptor_type;
                    VAR data   : data_buffer;
                    VAR dlen   : integer;
                    VAR flags  : flags_type;
                    VAR opt    : opt_type;
                    VAR result : integer );
```

Parameters:

cd (input)

Connection descriptor identifying the connection endpoint. Since a connection cannot be shared for receipt of data, the first word of the connection descriptor must not contain the pin of the calling process. (See IPCSEND.)

data (input/output)

During connection establishment, this parameter is not used.

On an established connection, this array is either the buffer where the data is to be placed or a list of addresses indicating where the received data is to be scattered. If flag bit #31 is set, then the data will be scattered (vectored), and the user is expected to supply one or two vectors as input. These specify where the data is to be placed. A vector consists of four words and has the following format:

```
 ------------
|    TYPE    |   Type:
|------------|        0 - Address is stack relative
|    DST     |        1 - Address is relative to a data
|------------|            segment index as returned by
|   OFFSET   |            the GETDSEG intrinsic. The
|------------|            user must be privileged.
| BYTE COUNT |        2 - Address is relative to the
 ------------             specified DST. The user must
                          be privileged.

                 Offset:
                          A DB-relative byte offset for
                          type 0, otherwise relative to
                          start of DST
```

Byte Count:
The length of the user buffer

Note that if the data is to be vectored, the 'data' array must contain exactly 8 or 16 bytes, and the 'dlen' paramter must be either 8 or 16.

dlen (input/output)

If receiving a response to a connection request, this parameter is not used.

If receiving data on an established connection, this parameter is bot input and output. On input, it gives the maximum number of unvectored bytes the user is willing to receive. This value must be greater than zero and no larger than 30,000. For vectored data [illegible] specifies the length of the vectors (8 or 16). On output, dlen [illegible] indicate how many bytes were actually received if waited I/O was used. For nowait I/O, the actual byte count will be placed in the tcount parameter of IOWAIT.

flags (input/output parameter)

Option flags This parameter is not required. Defined flags are

flags [no_output_flags] (bit #16, input)

If this bit is set and the intrinsic is called in nowait mode, the flags parameter will not be updated upon completion of the intrinsic. This allows a calling procedure to have a local flags parameter and still complete before the IPCRECV completes.

flags [discarded] (bit #[illegible] output)

This flag is used only on the ipcrecv following an ipcconnect. It indicates that some of the call user data returned with the connection request reply message had to be discarded because the user's buffer was too small. Note that all flags are optional, so if the user has not specified a flags parameter in the IPCRECV call, the RESULT parameter must be examined for a value of 142.

flags [more_data] (bit #26, output)

In general terms, this bit is intended to indicate that there is (or may be) more data to be received after the completion of the IPCRECV. This bit gets set when:

a. The TCP protocol is operating in stream mode. The assumption here is that there could always be more data.

b. A message was received which was larger than the user chose to accommodate. In this case, the remaining data will be available in the protocol module's buffer and can be read with another IPCRECV if the destroy_data flag was not set.

For connection completion, this flag bit is not used; if call user data was received and the user buffers could not accommodate all the data, then the discarded flag will be set.

If flags bit #16 is set when the intrinsic is called, the more data flag will not be presented to the user if nowait mode is being used.

flags [destroy_data] (bit #29, input)

With this flag, the user can direct IPC to throw away any data remaining after the user's buffers have been filled. The only way the user will know that data has been discarded is that in message mode (TCP), the more_data flag will be set upon completion of the IPCRECV. In stream mode, there is no mechanism for the user to detect this. Therefore, it is recommended that this flag not be used in stream mode. This flag is not used during connection initiation.

flags [preview] (bit #30, input)

If set then the user can preview the received data. This means that the data can be obtained from the 'data' array, but will not be removed from the protocol module's buffer. This flag should be mutually exclusive with the destroy_data flag, and if the user sets both then an IPC error will result. This flag is used only on an established connection.

flags [vectored] (bit #31, input)

If set then the data is to be scattered. This means that vectors must be supplied on input in the 'data' array to indicate where the received data is to be placed. If bit #31 is not set, the received data will be placed directly into the 'data' array. A maximum of two vectors may be provided.

This flag also selects vectored/non-vectored for call user data. However, in this case, the vectors are placed in the opt array.

opt (input/output)

Array of options, assembled with an ADDOPT intrinsic [illegible]efined are:

call_user_data_recv (code=5, len=n, n byte buffer) (output)

This option specifies that call user data may be received during the connection establishment. The data can be either vectored or non-vectored, depending upon the state of flags bit 31 when the intrinsic is called. See the above discussion of the 'data' parameter for a descripton of the vector(s). The maximum non-vectored length is 512 bytes. Non-vectored data is placed in the opt entry area, whereas vectored data is put into the buffers specified by the vectors. Therefore, if nowait I/O is desired and the call user data buffer (which is part of the opt record) is to be released after the initial call to IPCRECV, vectored data should be selected. Otherwise, the location previously held by the call user data buffer will be overwritten with the data, possibly creating undesirable results.

The actual byte count received is available to the user. If the data is not vectored, then the byte count will be placed in the length parameter of the option entry and may be determined with a READOPT or by explicit knowledge of the location of this parameter within the opt array. If the data is vectored, then the location of the received count depends upon whether nowait I/O is being used. If waited, then the count is put in the length parameter of the first vector (which is assumed to still be in its original place in the opt entry). This will be the number of TOTAL bytes received, even if there were two vectors. If nowait I/O is selected and the data is vectored, then the total byte count will be available in the tcount parameter of the IOWAIT.

If not enough buffer space was allocated for the actual amount of data received, then the discarded flag will be set. However, the flags parameter is optional and if it is not given in the initial call to this intrinsic, there [illegible] be no indication of discarded data.

data_offset (code=8, len=2, 2 byte integer, input option)

Defines a byte offset from the data parameter's address where IPC is to begin placing the received data. This opt must not be used if the data is to be vectored!

protocol_flags (code=144, len=4, 4 byte buffer) (output)

This opt contains 32 bits of protocol-specific flags. Currently, only X25 uses these flags, although IPC does no protocol checking on this opt. Undefined flags cause a SOCKERR 155. If bit #16 of the flags parameter is set and nowait is being used, the protocol flags will not be updated upon completion of IPCRECV. The only flags defined for this intrinsic are:

d_bit_flag (bit #18) (output): Indicates the state of the D bit in the X25 packet. The D bit is used to specify end-to-end acknowledgement of the data. This bit is not used by IPCRECV for connection establishment. (There is no D bit for call user data.)

q_bit_flag (bit #19) (output): Indicates the state of the Q bit in the X25 packet received by the protocol module. During connection establishment, IPCRECV does not change this bit. On an established connection, this bit specifies that the data is control information intended for a pad.

urgent_data_flag (bit #27) (output): This flag is used only on an established connection and indicates that urgent data has been received. This bit is not output to the user if flags bit #16 is set when the intrinsic is called in nowait mode.

result (output)

Indicates whether the request was successful. If nowait I/O is used, the result parameter will give information about the initial call to the intrinsic, but the parameter will not be updated upon final completion of the request. To examine the result of the IPCRECV completion, the user can call IPCCHECK.

Discussion:

The IPCRecv intrinsic serves two purposes: 1) to receive a response to a connection request, and 2) to receive user data on a connection. When receiving data, a user can choose to preview the data and/or recieve it into user buffer(s).

In receiving a response to a connection request (a call to IPCConnect), the intrinsic returns nothing in the data buffer. A result of zero indicates a successful connection establishment. Various error codes indicate unsuccessful establishment. One such error code will indicate rejection by the destination. Call user data if available will be returned in the buffer provided by the option call_user_data_recv or into user buffers if vectored data is selected.

When receiving data, the user will be waited until some data arrives (or a timeout occurs). The dlen parameter will reflect how much data was received. If there is more data than was requested, the more_data bit will be set and the remaining data can be received with the next call to IPCRecv. A user will never receive any data beyond an end of message marker with a single call to IPCRECV.

3.3.11 IPCGIVE

Function:

To give a socket or connection endpoint to another process.

Declaration:

```
PROCEDURE IPCGIVE (     descriptor : descriptor_type;
                    VAR give name  : socket_name_type;
                        nlen       : integer;
                    VAR flags      . flags_type·
                    VAR result     : integer
```

Parameters:

descriptor (input)

Socket or connection descriptor of entity to be passed

give_name (input/output)

Socket name to be temporarily assigned to the socket or connection to be given away. This value must be matched by the user attempting to get the connection/socket. If the user specifies a length of zero for this name, an eight byte value will be randomly assigned and returned in this parameter. If the name is supplied by the user, it must be no more than sixteen bytes.

nlen (input)

Byte length of the specified name. This value may be zero indicating the IPC facility is to assign the name.

flags (input/output)

Option flags. No flags are currently defined.

result (output)

Resultant error code, else zero.

Discussion:

The IPCGIVE intrinsic is used to pass a socket or connection to another process. A name will be associated with the connection/ socket which must be matched by the process trying to receive the connection/socket. This name can either be specified by the user or assigned by the IPC facility. This name will be temporary (until the connection/socket is taken or destroyed) and can only be referenced by the IPCGET intrinsic (not by IPCLOOKUP).

The syntax of the of the name is the same as for the other socket intrinsics permitting names (see Section 3.2.1). This allows users to use a socket's well known name for the IPCGive and IPCGet intrinsics.

Once this intrinsic has been invoked, the user no longer has access to that socket or connection descriptor. If a process expires after giving away a socket/connection but before another process receives it, the connection or socket will be destroyed. It should be noted that some systems may wish implement a means for a process to give away a socket/connection and expire without destroying the socket or connection.

Users may continue sending data to a socket or conection while it is being given away. It is the user's responsibility to notify other users that a socket/connection has been given away, and what name has been assigned for retrieving the socket or connection.

3.3.12 IPCGet

Function:

To receive a connection endpoint or socket which has been given away.

Declaration:

```
PROCEDURE IPCGet ( VAR give_name  : socket_name_type;
                       nlen       : integer;
                   VAR flags      : flags_type;
                   VAR descriptor : descriptor_type;
                   VAR result     : integer );
```

Parameters:

give_name (input)

   Name assigned to the socket or connection when it was given away.

nlen (input)

   Length, in bytes, of specified name.

flags (input/output parameter)

   Option flags. None are currently defined.

descriptor (output)

   Connection or socket descriptor for socket/connection received.

result (output)

   Resultant error code, else zero.

Discussion:

The IPCGet intrinsic is used to take a connection or socket which has been relinquished via the IPCGive intrinsic. The name identifies

3.3.13 IPCCONTROL

Function:

Performs special operations.

Declaration:

```
PROCEDURE IPCCONTROL (      descriptor  : descriptor_type;
                            request     : integer;
                        VAR wrtdata     : data_buffer;
                            wlen        : integer;
                        VAR readdata    : data_buffer;
                        VAR rlen        : integer;
                        VAR flags       : flags_type;
                        VAR result      : integer );
```

Parameters:

descriptor (input)

Either a socket descriptor or a connection descriptor.

request (input)

Defines what control operation is to be performed. See the discussion below for a list of the defined requests.

wrtdata (input)

Byte array used to present any input data. For certain requests, wrtdata will contain the actual data, whereas other requests allow list of addresses (vectors).

wlen (input)

Byte length of the wrtdata array.

readdata (output)

If the request results in data being returned to the user, this parameter is the destination.

rlen (input/output)

On input, used to specify the maximum amount of data the user is willing to receive. On output, tells the user how much data actually was received. See the various requests for details.

Option flags. 32 bits, each selecting an option. Defined is:

flags [vect/transtrace] (bit #31, input)

This flag bit has a dual usage, depending on the request code. If the request is to enable IPC trancing, then this bit is used to select whether transport tracing should also be enabled. If the request is to accept or reject a deferred connection, then the bit is used to select vectored data (data which will be gathered from user buffers).

result (output)

Indicates the result of the request.

Discus [illegible]

The IPCCONTROL intrinsic is used to perform special requests on sockets. A request can include receiving [illegible]ormation about a socket. The currently defined control functions are:

1 - E[illegible] nowait (asynchronous) I/O for the specified socket or co[illegible]ection. (Uses Descriptor, Request, and Result.) If this request is selected, then the user's proc[illegible]ss can continue its acti[illegible] while the I/O intrinsic waits for the requested transfer to comple[illegible] Operations such as IPCSEND, IPCRECV, and [illegible]RECVCN will not actually complete until the user calls the IOWAIT [illegible]insic.

2 - [illegible]rm waited (synchronous) I/O for the specified socket or connection. (Uses Descriptor, Request, and Result.) This means that the calling process will wait for the intrinsic to complete the operation before ontinuing. If the user tries to switch from nowait I/O to waited I when there is uncompleted I/O, sockerr 71 will be returned. Also, if the user tries to enable waited I/O when software interrupts are enabled, error 112 will be returned. (enhancement in near future)

3 - Allows the user to change the default timeout for receives. The wrtdata array must contain 2 bytes of timing value in tenths of seconds. A zero time value turns off receive timeouts. The default timeout will be sixty (60) seconds. The maximum time is 3,276.7 seconds. If a larger value is requested, error 76 is returned.

When an IPCRECV is called by the user, the timer is set to the value specified. [illegible] the [illegible]PCRECV completes before the timer pops, then the timer is aborted.

If a timeout occurs before the receive intrinsic completes. the result parameter of the IPCRECV will be updated to show error #59, and the dlen parameter will be set to zero. (This is true only for nowait I/O.) The pending receive will be terminated , but the connection, if established, will not be closed.

9 - ACCEPT_DEFER_CONN. This request tells the protocol module to accept a deferred connection. The source socket which received the call must be in deferred call acceptance mode, or error #166 is returned.

The user may send call-related data along with this message to the protocol module. If the wrtdata array is specified with this request, then the contents of the array will be interpreted as "call user data". The format of the wrtdata array is the same as the opt parameter of other intrinsics and must be specified with an INITOPT/ADDOPT sequence. The call user data may be vectored If flags bit #31 is set for this request, then the wrtdata array is assumed to contain one or two (maximum) vectors which point to user buffers. For this request, the wlen parameter is not used.

10 - RESET_VC. This request causes X25 to send a reset packet on the virtual circuit associated with the connection socket. It is only valid on connection sockets and only for X25. Wrtdata may contain the cause and diagnostic fields for inclusion in the reset packet. Wlen must be 2. No readdata is associated with this request.

11 - INTERRUPT_VC. This request causes X25 to send an interrupt packet on the virtual circuit associated with the connection socket. Wrtdata must contain 1 byte of user data to be put in the interrupt packet user data field. This request is only valid on connection sockets and only on X25.

12 - WHY. This request returns the reason for the IPC error or event on an X25 connection. The readdata parameter is required, and rlen must be 4. The first byte of readdata contains the type of packet that caused the error (reset, clear, restart) or the unsolicited event (interrupt). If the type is reset or clear, the third and fourth bytes will contain the cause and diag bytes from the packet (the second byte will be zero). If the event was an interrupt, the second byte will contain the interrupt code from the packet, and the last two bytes will be zero. This request is only valid on an X25 connect socket.

Note that the WHY request is only useful if the user needs to obtain the data associated with the event; the type of event is indicated by the error code returned. For example, if a SOCKERR #146 occurs, the user knows that a reset packet was received. An IPCCONTROL will be necessary only if the cause is of interest.

13 - NO_[illegible]TY_TIMEOUT. This request sets the no activity timeout value (X25 o[illegible]). if no user generated activity occurs on the connection for this amount of time, then the connection is automatically cleared and an error is returned on any subsequent IPC routine call. The user must use the IPCSHUTDOWN intrinsic to remove the connection socket. Wrtdata must contain a 16 bit integer representing the timeout value in minutes. If the value is equal to zero, the timer will be disabled. Wlen is 2. Readdata is not used for this request. This request is only valid on connection sockets. A default timeout value is def[illegible]d at configuration time.

15 - REJECT_DEFERRED_CONNECTION. This request is used to reject a connection request which was previously deferred. The connection must be in the vc wait confirm state, otherwise a SOCKERR #166 will be

returned. If the wrtdata array is given, then call-related data will be sent back to the protocol module (and presumably back to the requesting node). The format of the wrtdata array is the same as for the opt array used by other intrinsics, and it must be initialized with an INITOPT/ADDOPT sequence. The call user data can be vectored by setting flags bit #31 and putting one or two vectors in the wrtdata array. The wlen parameter is not used.

256 - Enable nowait receives/disable nowait sends.

257 - Enable nowait sends/disable nowait receives.

258 - ABORT OUTSTANDING NOWAIT RECEIVES. The connection is not aborted.

259 - ENABLE USER TRACING. This request enables tracing for a socket and possibly also for the protocol module. The wrtdata array can contain up to three optional entries for the tracing. This array must be initialized with an INITOPT/ADDOPT sequence. The three available opt entry codes are:

   131 - Specify trace file name. The data in the wrtdata entry contains the name of the trace file to be used. The name length must be greater than zero and less than 36.

   132 - Specify the number of logical records in the trace file. The wrtdata entry must be two bytes, giving a 16-bit number of records.

   133 - Specify the maximum number of user bytes to be traced. The wrtdata entry must be two bytes with a value no larger then 8192.

   The readdata array if specified will return the actual name of the trace file, including the group and account. If the user specified the file name, then the current group and account will be appended. If no user file was specified, then one will be created. The file name so created will be of the form SOCK????, where ???? is four random digits.

   If bit #31 of the flags parameter is specified with this request, then protocol module tracing will be enabled along with user data tracing. (This is not allowed for a TCP call socket.)

260 - DISABLE TRACING.

261 - ENABLE_IMMEDIATE_ACK. This request instructs the TCP protocol module to acknowledge received frames immediately.

262 - ENABLE_SEND_TIMEOUT. Sets a timer for connection send operations. The wrtdata array contains the timeout value in tenths of seconds and must be exactly two bytes in length. A time value of zero will disable the timer. The default is no send timeout. If the user tries

to set a timeout on a connection which is being shared, a SOCKERR 167 will occur.

512 - ALLOW_SHARED_CONNECTION. Allows other processes to share the connection for sending data. An error will be returned if the descriptor is not a connection. (Call sockets cannot be shared.) Also, if a timer has been enabled for sends on the connection, an error will be reported. There can be a maximum of eight shared connections per process. This request is available only to privileged users.

513 - ENABLE_SOFT_INTERRUPTS. This request is used to enable or disable software interrupts on the socket. The wrtdata array should contain two bytes which define the user's plabel. A plabel of zero will disable software interrupts. If there is any I/O outstanding, the request to enable software interrupts will be denied. Privileged users only.

A future enhancement will disallow software interrupts with waited I/O and will also not allow software interrupts to be disabled with I/O outstanding.

514 - RETURN_SOCK_ADDRESS. For privileged users, the specified socket's address will be returned in the readdata array. The readdata array should be at least six bytes long to accommodate the returned string. The rlen parameter is not used as an input, but will be updated on output to indicate the actual length of the address. The returned address has the following meaning:

DESCRIPTOR TYPE	ADDRESS MEANING
call socket	port address of socket (for TCP, len = 2 bytes)
connection from IPCCONNECT	local port address of connection socket (for TCP, len = 2 bytes)
connection from IPCRECVCN	remote port address of connection socket in bytes 0 and 1; remote internet address of node in bytes 2 through 5

515 - SET_TCP_WINDOW_PARMS. This request is available to privileged users only and is only valid for connections (not call sockets) using TCP. Various parameters which control the sending and receipt of TCP messages can be altered with this request. The wrtdata parameter contains a code for the specific parameter to be altered. The wrtdata array is formatted with an INITOPT/ADDOPT sequence. Request 515 is

intended for use with TCP message mode only. The supported opt codes are:

3 - Maximum send message size in bytes. The wrtdata entry must contain 2 bytes in the range 1 to 32,000. If the value is smaller than previously set, the request will be ignored. The default is 1024.

4 Maximum receive message size in bytes. The wrtdata array must contain 2 bytes in the range 1 to 32,000. If the value is smaller than previously set, the request will be ignored. The default is 1024.

134 - Maximum send burst The wrtdata parameter must contain two byes which specify a number in the range 1 to 7. This number sets the number of messages that can be pipelined to the other end of the connection without the messages necessarily being processed by the peer. A user can continue sending messages without forcing the peer to process them, if the number outstanding messages is smaller than the burst size and there is sufficient window space. The default burst size is 3.

135 - Maximum receive burst. The wrtdata parameter must contain two bytes representing a number in the range 1 to 7. This is the number of messages which can be pipelined to the receiver's end of the connection without being processed. This burst size is used to calculate the window which TCP will be advertising. That is, the window is the maximum receive size times the receive burst size. The default receive burst size is 3.

136 - Window threshold. The wrtdata array contains two bytes which represent a number in the range [illegible] to 100. This is the percentage of the total window that must be utilized before sending a window update packet to the remote peer. It is used to prevent TCP from generating packets merely for updating the window. However, packets for piggybacked updates will continue to be sent. The default window threshold is 50%.

The IPCCONTROL intrinsic is "option variable". That is, the number of parameters actually supplied in the intrinsic call is variable. The request code and the descriptor must always be supplied, but other parameters may not be required for the specific request.

This intrinsic cannot be called in split stack.

3.3.18 IPCSHUTDOWN

Function:

To release a call socket, destination descriptor, or connection descriptor. Associated resources are also released.

Declaration:

```
PROCEDURE IPCSHUTDOWN (     descriptor : descriptor_type;
                        VAR flags      : type_flags;
                        VAR opt        : type_opt;
                        VAR result     : integer );
```

Parameters:

descriptor (input)

Either a socket descriptor, connection descriptor, or destination descriptor.

flags (input)

32 bits of optional actions. The only defined flag is:

flags [graceful_release] (bit #17, input)

If this flag is set, the connection will be gracefully released.

opt (input)

Array of options, initialized with an INITOPT/ADDOPT sequence. Defined is:

reason_code (code=143, len=2, 2 bytes, input)

This option allows the user to specify two bytes of information about the shutdown reason. The reason code is only allowed for X25 [illegible] or any other protocol, a sockerr #145 will occur. The bytes are placed in the cause (first byte) and diagnostic (second byte) fields of the X25 clear packet. The reason information may be supplied only for a connection socket. Otherwise, a sockerr #8 will occur.

result (output parameter)

Resultant error code, else zero.

Discussion:

The descriptor is the only required parameter.

This intrinsic may not be called in split stack.

This intrinsic permits a user to close a socket or release a connection. If a call socket is being shut down, users may continue using any associated connections which have been established. The effects of shutting down a call socket are:

1. Any timers set on the socket are aborted.

2. If software interrupts were set for the socket, they are disabled.

3. If there are any pending connection requests on the socket, the requests are rejected.

4. Any names associated with the socket are removed.

5. If tracing is enabled for the socket, the trace file will be closed.

6. If logging is enabled on the socket, the closure will be logged in the active NMLG file.

The effects of shutting down a connection are:

1. If the connection was being shared by several processes, error messages are sent to the other users. This causes any outstanding requests on that connection to immediately complete.

2. If software interrupts were enabled on the connection, they will be disabled.

The graceful release capability is intended to allow closing a connection without loss of inbound data. When one node initiates a graceful release, a message is sent to the remote node informing it of the event. The connection will then go into a simplex state with the initiating node being able to receive but not send. Therefore, if data is in transit to the initiating node, it will not be lost. The remote node must at sometime call IPCRECV to know that this has happened. The connection will remain in a simplex state until the remote node initiates a graceful release or until the local node calls IPCSHUTDOWN without the graceful release option.

A sockerr #102 will result if graceful release is selected and any of the following conditions exist:

1. The connection is in the vc'wait'confirm state. That is, a connection request has been received, but the connection has not been accepted.

2. The connection is in the vc'simplex'in state. This could happen, for example, if an established connection has already been gracefully released.

3. The connection is in the vc'connecting state. In this case, a connect request was issued, but the connection is not yet established.

4. The connection has been aborted, possibly due to an irrecoverable error.

5. The pending outcount on the connection is not zero. For example, IPCSEND was called in nowait mode and has not completed.

6. The protocol module does not support graceful release.

3.3.19 IPCDest

Function:

Creates a destination descriptor.

Declaration:

```
PROCEDURE IPCDest (     socket_kind  : integer;
                    VAR location     : location_type;
                        location_len : integer;
                        protocol     : integer;
                    VAR proto_addr   : packed array of bytes;
                        addr_len     : integer;
                    VAR flags        : type_flags;
                    VAR opt          : type_opt;
                    VAR dest_descrip : descriptor_type;
                    VAR result       : integer );
```

Parameters:

socket_kind (input)

Defines the type of socket. Refer to the IPCCREATE discussion for a of list socket kinds. There is no default.

location (input)

Name of the node on which the remote socket resides. This parameter may be omitted, in which case the location is assumed to be local. If omitted, then the location_len parameter must also be omitted.

location_len (input)

Byte length of the destination node name. If this parameter is given, then the location parameter must also be given. However, the location_len may be zero, indicating a local destination (loopback).

protocol (input)

Defines the protocol used by the remote socket. Refer to the IPCCREATE discussion for a list of valid protocols.

proto_addr (input)

Protocol relative address which will be associated with the destination descriptor. For non-privileged users, the address value must be in the range %74057 to %77777. Otherwise, sockerr #164 will occur.

addr_len (input)

Byte length of the protocol address, if given. For privileged users, the length may be no less than one and no greater than 16 bytes. For non-privileged users, the addr_len must be 2 bytes. Otherwise, sockerr #165 will occur.

flags (input/output)

Option flags. No flags are defined for this intrinsic. If any flags are given, a sockerr #7 will occur.

opt (input/output)

Array of options. None are defined for this intrinsic. If the opt parameter is given, then its length must be zero. Otherwise, sockerr #8 will result.

dest_descrip (output)

Destination descriptor. Value returned which is to identify the destination socket.

result (output)

Resultant error code, else zero.

Discussion:

This intrinsic may not be called in split stack.

The required parameters are socket_kind, proto_addr, addr_len, and dest_descrip.

The IPCDEST intrinsic is an alternative to the IPCLOOKUP intrinsic and allows the user to create a destination descriptor which can be used for establishing connections and sending data.