DATE: March 19, 1981

TO: R & D Personnel

FROM: Tracy Taylor

SUBJECT: SUBSYSTEMS PROGRAMMING GUIDE TO NPX

REFERENCE: PE-TI-643

KEYWORDS: None

ABSTRACT

This informal PET describes the What, when, and How of using NPX in PRIME subsystems.

It is intended to be a reference for those engineers using NPX to distribute their subsystem over PRIMENET.

|Changes in this revision are marked with revision bars. The major |changes are:

| o calls that previously had no return codes now have them

| o maximum concatenated output length has been reduced to 8 KB

| o an experimental asynchronous interface is specified

Table of Contents

Introduction

In a single system we invoke many services (to access system resources) by a subroutine call either into PRIMOS or a shared library. NPX (in the form of remote procedure call) provides a mechanism for invoking services when the service is to be performed on a remote node linked by PRIMENET.

For example, a call to T$MT is used to manipulate a magtape drive on the system where the call is made. A remote procedure call to T$MT from a different system would allow access to that tape drive from a remote system though PRIMENET.

In general, NPX can be used to access objects normally accessed via the PCL mechanism from a node where the object does not reside. This object abstraction can be applied to such varied objects as physical devices, MIDAS files, queue files, in-memory data bases, electronic mail data bases, shared programs etc.

1 When is it appropriate to use NPX?

The objects and subroutines that access them must meet certain criteria to be good choices for NPX-distributed applications. The following criteria are written as to be applied to a subsystem that you are considering distributing.

1.1 Dynamicllly linked

The subsystem must be accessed via dynamically linked subroutines. This means either a system gate or a shared library.

1.2 Remote object

The subroutine interface must have an object descriptor (explicit or implicit) for an object or resource that can be considered to be local or remote. (Alternately an object could be known to always be remote, for instance, a dedicated mail machine.)

1.3 No common blocks

The user and the subsystem in question may not use common blocks to communicate. The subsystem may use common blocks to communicate between its own internal subroutines. Common blocks may not be used to communicate between unrelated distributed subsystems.

1.4 Frequency of calling

The interface that is remotely called must have a reasonable frequency of calling. For example in a distributed compiler, a remote procedure call for each source file disk-record might be reasonable, but a remote procedure call for each token would not be. Subsystems can often be broken at several different points; it may take some effort to partition an existing subsystem in a reasonable manner and keep the size and amount of communication traffic to a minimum.

1.5 PL/1 conditions

The slave contains an ANY$ handler. Any condition which is not handled by the target subroutine will cause the NPX_SLAVE_SIGNALED$ condition to be signaled on the users stack. This condition is non returnable. Information about the original condition is available. See (PRIMOS>R3S>DF_UNIT_.PLP as an example).

1.6 No terminal I/O

No terminal I/O can be done inside the remote subsystem...any I/O from or to the user terminal must be done locally and passed as an argument across the R$CALL interface to the remote subsystem.

2 Functional Specification

2.1 Functional Description

NPX allows a user process on one system to call any dynamically linkable subroutine on any remote system enabled for this facility. The local process (the "master") acquires exclusive use of a server process (the "slave") on the remote node where the procedure call is to be executed (the "target node"). For example, a program running on ENB could call the MIDAS routine NEXT$ for remote execution to read the next MIDAS record from an ISAM file on END. The master-slave relationship is retained until explicitly broken by the master, logout, or unrecoverable node or communications failure.

The NPX mechanism is intended to support up to 15 arguments in each remote call. Arguments are copied from the calling (master) process to the server process where the target call is made. Return arguments are copied back to the caller in an analogous fashion. The maximum length of all arguments concatenated together is 8k bytes due to ring0 stack size limitations. Each argument may be an input argument, an output

argument, or both. Pointer or LOC() arguments will be supported but some special rules will apply to inform the NPX mechanism of the memory size of the referenced argument. INTEGER*2, INTEGER*4, aligned char(*), aligned char(*) varying, and pointer, both scalar and aggregate data can be passed through the NPX mechanism. Each argument to a local subroutine maps to a triplet of arguments for NPX. The triplet includes the argument itself, its length, and keys describing its data type.

The return lengths of aggregate arguments can be specified by other arguments set as a result of the target call. For example in PRWF$$, argument 6 (the returned data length) specifies the return length of argument 3 (The returned data).

Static external storage will remain valid in the slave's address space across individual remote calls. This allows intermediate results to be retained in the slave.

To reduce resource overhead, unbound servers will wait in a general pool with a minimum of active segments and no wired memory. All file units will be closed, and any additional segments allocated to a server will be returned to the system when a slave is released and returned to the pool.

Five shared subroutines will be supplied to use the NPX facility: R$ALOC(), R$CALL(), R$WHER(), R$CVT(), and R$RLS() (see Section 3 for details on the calling sequences of these subroutines).

NPX will have several robustness features to insulate users from transient network failures:

It will recover from X.25 resets in a manner transparent to the user. (These can occur every 5 minutes to once a week.)

Each slave will have a user setable timer telling the slave how long to wait (up to 511 minutes) for the master to reconnect after the virtual circuit is cleared due to such events as halt-warm start in the master's system or public data network (PDN) congestion. If the slave times out he cleans up and returns to the free server pool.

A key will be provided to allow retries to obtain a server on a remote node if none are initially available. This "wait for a server" option is user quitable.

ACLs will be used to restrict the scope of the slaves access on target systems. The slave will inherit its master's user name, project(s) and node name. This gives control of remote resources to

the administrator of the target system (the system that owns the resources) not the master's system administrator (the system that wants to consume or use them).

The virtual circuit from the master to the slave will be a secure one, this enables PRIMOS to guarantee the identity of the master and use ACLs on the remote system.

2.2 Areas That Are Likely To Get Addressed After the First Cut

Since some (non domestic) PDNs charge on a per-virtual circuit and per- hour of connect time basis some interest is being generated in multiplexing several VCs over one (or two) VCs. This seems to make most sense for classes of users (e.g., remote login users might be multiplexed on one VC and NPX users on another) as each user might be expected to trust the other users in his class and know their pattern of traffic.

2.3 What NPX Will Not Do

The following features may not ever be implemented, and this should not be viewed as commitments for the future.

Allow passing of data between caller and callee through static external memory (e.g., network page faults).

Support a "keep this argument in the server's space for later reference" argument type.

Allow each shared library to allow or deny access to its contents to NPX slaves on a per-node basis.

Support label or altrtn arguments to calls.

Support terminal I/O from the slave.

Allow quits to be passed from the master to the slave in order to stop the slave's activity and return control to the master.

Have alternate network-path protection implemented via NETCFG. That is if multiple paths exist to a node then NPX to that node will be allowed or disallowed for all paths as a group, not individually.

3 The NPX Subroutine Interface

Five shared subroutines will be supplied to use the NPX facility.

R$CVT (NODENAME, NODENAMELEN) /* returns I*2 */

To obtain a node number from an ASCII name use the function R$CVT which returns an integer value.

NODENAME The name of the node (e.g., 'ENB').

NODENAMLEN (I*2) Length of the nodename in bytes.
|The octal value 100000 (the largest negative number) is returned if thi
|s
|node is not known.

|R$WHER (KEY, OBJECT NAME, OBJECT_NUMBER, CODE) /* Returns I*2 */

|To find what node an object which you wish to access using NPX is
|physically located on use the function R$WHER.

|KEY tells R$WHER what kind of object is being located.

|Current KEYS are:

|K$NAME return the node number of the file path name specified in
| OBJECT_NAME

|K$UNIT return the node number of the file unit specified in
| OBJECT_NUMBER
|OBJECT_NAMEASCII name of an object to be located char (128) VAR.
| (e.g. file name)

|OBJECT_NUMBERFor numeric objects (I*2) (e.g. file unit)

|R$ALOC (NODENUM, CODE)

Since many subsystems use the same slave, a global per slave allocation count must be shared by all subsystems sharing a slave. To allocate a slave for a specific node R$ALOC merely increments the per node counter, no guarantee is made as to whether or not a slave is actually available on a target node. Calls to R$ALOC and R$RLS must be paired analogous to quit inhibit calls in order to maintain the counter correctly.

|R$RLS (NODENUM, CODE)

Decrements the use count for this slave and releases the slave if the count reachs 0. In this case the server returns to the pool of available servers.

R$CALL(RKEY, NODENUMBER, PROCNAME, PROCNAMLEN, RCODE,
ARG1, ARG1LEN, ARG1TYPE, ARG2, ARG2LEN, ARG2TYPE, ... ,
ARGn, ARGnLEN, ARGnTYPE)

Calls the target procedure on the specified node. At least one R$ALOC call must have been made already for this node. Arguments to the target procedure are specified in triplets as specified below:

RKEY (I*2) key to NPX (see below)

NODENUMBER (I*2) NPX node number to activate a server on. (See R$CVT to obtain node number from node name.)

PROCNAME ASCII name of the target (called) subroutine. Under current search rules this must be dynamically linkable

PROCNAMLEN (I*2) Length of the target subroutine name in bytes

RCODE (I*2) Return code for the "remoteness" of this call, not the code from the target subroutine

ARGn (any type) Nth argument to the target subroutine

ARGnLEN Length of the Nth argument in its basic units (ie., bytes, words, double words or quadwords

ARGnTYPE (I*2) Bits which describe the data type, direction, and indirection of the nth argument. All keys are additive.

TYPE

K$FB15, K$I2 Argument is fixed bit 15

K$FB31, K$I4 Argument is fixed bin 31

K$FL Argument is float bin (2 words)

K$DFL Argument is double floating bin

K$CHAR Argument is a fixed length character string (must be aligned)

K$VCHR Argument is a PL1 character string of varying length (must be aligned)

DIRECTION

K$IN Input only argument...the target subroutine uses this argument as an input parameter.

K$OUT Return or output argument...the subroutine sets the value of this argument for use by the caller. K$IN and K$OUT can be added together to specify that an argument is both passed to

and returned from the target subroutine.

K$REF + m — Refer to argument number m after the target call is made to obtain the length to return. The length will be an I*2 number representing n units of the data type of the refered to argument. ARGnLEN should be the maximum possible length of the pointed to data structure (this is used for the slave's allocation temporary storage). (e.g., if the referred to argument is K$CHAR, then the refer value will be in bytes, if the referred to argument was K$FB15, then the refer length would be understood to be words. K$REF requires that K$OUT be used.

INDIRECTION

KPTR, KLOC — Argument is a pointer or LOC() variable. In this case, argument length is the length of the structure pointed to in 16 bit words. If the argument is also REF, the arguments return length will be in its own data type. (Only K$I2 works now).

KEYS

K$WFRC + n — The slave will wait n times 15 seconds for a reconnect dialog if the Virtual Circuit is cleared by network failure during a remote request. n can be 0 to 127. This key gets reset on every request.

K$FUNC — This is a function call, please return the L register value set by the target subroutine to the caller of NPX. Function values not returned in the A or L register are not returnable though NPX.

K$RTRY — Keep trying to get a server if none are initially available. Quit will cause an exit and print the message "no remote servers available" - ignored after the first request.

Possible return codes in RCODE

0 — Operation complete.

E$RLDN — Server's system or link has gone down since the users last remote request. This request has not been started. If K$WFRC is used, the user will wait n minutes in the abortable reconnect state before returning this code.

E$FONC — Servers system or link was lost in mid request. The link could not be reestablished in the allowed time.

The target call may or may not have been executed.

E$UNOP Server was lost due to a cold start or force logout or server timeout after virtual circuit clearing) since last request.
(The VC was reestablished but the slave no longer existed.)

E$FABT Server error.

E$NRIT Remote subroutine linkage not permitted.

|E$NSLA No servers available in the time allowed to get one.

E$IREM No NPX calls allowed to that system from this system.

E$NETE An uncorrected Low level network error has been detected by R$CALL.

E$FIFC OLDFAM() has been called with a bad famcode.
(Internal error.)

|E$PNTF (target) procedure not found.

Hence a PRWF$$ call to read 10240 words on file unit 35 looks like:

```
RNODE =  R$CVT('ENB',3)              /* GET THE NODE NUMBER */
CALL R$ALOC(RNODE)                   /* ALLOCATE A SLAVE FOR THAT NODE */
   .
   .
   .
CALL R$CALL (0,RNODE,'PRWF$$',6,RCODE,
+            K$READ+K$POSR,1,K$IN+K$FB15,  35,1,K$IN+K$FB15,
+            LOC(BUF),10240,K$PTR+K$FB15+K$REF+6,   10240,1,K$IN+K$FB15
+            0010240,1,K$IN+K$FB31,        NUMRED,1,K$FB15+K$OUT,
+            CODE,1,K$OUT+K$FB15   )
```

Note that the third argument triplet uses the K$REF key so that the number of storage units (of type K$FB15) actually read will be returned in BUF.

4 Coding rules and tricks

4.1 Implied context & variable length structures

NPX is an easy way to access PCL based resources, however some interfaces to these resources may be difficult to R$CALL efficiently due to such things as variable length return arguments where the length is not in the first word of a return argument (K$REF is unusable in this case).

One way to circumvent this is to insert a procedure that is called remotely only to set up the implied context before calling the subroutine that needs the implied context. For example:

```
X:  PROC(...);
    IF REMOTE THEN CALL R$CALL(...NODENUM,...'Y'...Z);


/* ON THE REMOTE NODE:     */

Y:  PROC(...);
    A=Z;
    CALL X(...);  /* CALL THE LOCAL X */
```

DBMS RAM depends on common blocks and MIDAS occasionally passes object lengths back in the second word of array arguments but this shouldn't keep these products from being distributed via NPX...tell me about the last "free lunch" you had.

4.2 Providing transparent remote access

Sometimes it is desirable to access remote objects just as if they were local...we do this with disk files via FAM today. Sometimes it is not desirable...remote login is an example of users deliberately and knowingly specifying a remote node. We have the opportunity to write our distributed subsystems in a transparent way when an object descriptor that can represent a local or remote object is passed thought the PCL interface to the subsystem. For instance the file system uses file units in this manner

The paradigm that we are currently pushing for remote access goes as follows:

```
      SUBROUTINE ACCESS(OBJPTR,RTNDAT)
      INTEGER OBJPTR,RTNDAT,NODE,R$WHER,JUNK,RCODE
      .
      .
      .
      NODE=R$WHER(K$type,OBJPTR,JUNK)
      IF (NODE.EQ.0) GOTO 10
      CALL R$CALL(0,NODE,'ACCESS',6,RCODE,
     1            OBJPTR,1,K$IN+K$I2,
     2            RTNDAT,50,K$OUT+K$VCHR)
      CALL ERRPR$(RCODE...)
      RETURN
```

```
10    CONTINUE
      .
      .
      .
```

Note that the subroutine access called its self on the remote node. This keeps the number of R$CALLs in prime software less than or equal to the number of user callable interfaces in that software.

4.3 File units

Any file units opened by the slave will be known only to the slave and not the file system on the master's system. File units opened in the slaves process space must be opened with the K$GETU key to prevent conflicts between your subsystem and any other subsystem using this slave on behalf of the (same) user.

5 Advanced Features

5.1 Static or Units for Slave Clean Up - LINE_DOWN$ condition.

In the event that the virtual circuit to the masters system is abnormally cleared (the only normal clear is on R$RLS) the slave will signal the condition LINE_DOWN$. If the user has made a static on unit for this condition in the slaves address space by R$ calling MKSOU$ or R$ calling a routine which calls MKSOU$ then this on unit will be invoked.

Virtual circuits are abnormally cleared when the physical link is broken, the master logs out, the slaves system halts and warm starts, or the master's system halts.

5.2 Asynchronous Remote Procedure Calls

Asynchronous NPX allows a remote procedure call to be broken into 2 separate operations. A Begin remote procedure call (R$BGIN) and an end remote procedure call (R$END). In effect, the first call sends the arguments to the slave and starts him running then returns. R$END is called to check the status of the outstanding call and optionally pick up the return arguments from the slave.

R$BGIN(KEY, NODE, PROCNAME, PROCNAMLEN, BUF, BUFLEN, RCODE,
ARG1, ARG1LEN, ARG1TYPE, ARG2, ARG2LEN, ARG2TYPE, ... ,
ARGn, ARGnLEN, ARGnTYPE)

R$BGIN has nearly the same calling sequence as R$CALL except that two new arguments are added between the PROCNAMLEN and RCODE (the fourth and fifth arguments of R$CALL). These arguments are a scratch buffer and buffer length for NPX. The size of the buffer in words is given by the formula

BUFSIZE = 100 + MAX (length of concatenated_input_args,
length of concatenated_output_args + 100).

NPX checks the size of the user supplied buffer against the the size of the buffer that it calculates is necessary, an error of E$BFTS is returned for buffer too small.

This buffer must exist for the duration of the remote P call. That is, if routine A calls B which calls R$BGIN and has Buf in its automatic storage then B can not return or release the automatic storage before the matching R$END.

currently only one remote procedure call may be pending per node at any one time. An attempt to queue R$BGIN calls for the same node will result in a return code of E$APND (Already PeNDing). In the future NPX may allow multiple pending calls per node or may block the caller of a second R$CALL or R$BGIN with an implicit call to R$END(...V$INFN...). R$END (KEY, NODE, TIME, BUFFER, CODE)

Checks the status of a single (or all) outstanding remote procedure calls. If a call is complete, the return arguments specified in the R$BGIN call are filled in.

KEY (I*2)

K$ANY Wait for, and pick up results from, any outstanding call. NPX returns the node # of the call that was completed in NODE.

K$SPEC Wait for or pick up the results from a specific node specified in node.

NODE (1*2) Under K$SPEC, the node number to check the status of. Under K$ANY, the node number (or 0) that just completed; examine CODE to see if this remote procedure call ended in an error.

TIME (1*2) Time to wait in tenths of seconds. Two special values are: V$INFN and V$NONE, infinite and no time respectively. If an operation completes before the timer runs out R$END returns. IF the timer expires without a call completing then E$SPND (Still PeNDing) is returned.

CODE (I*2) return code. This may be any code defined for R$CALL plus any of:

E$NPND No calls are pending (on specified node if K$SPEC)

E$SPND Still pending. The remote node is still working on the target call or data is in transit in the network.

0 Call complete for node # NODE. Results have been returned in to the R$BGIN K$OUT arguments.

6 Disclaimers

Asynchronous NPX is as yet unproven for general use...any engineers thinking about using it should contact the network group for consultation.

Two features are included in the specification that are not currently

in NPX. They are: K$WFRC and K$RTRY.