KEV 22-NOV-74 16:01 24575

ELF and ELF related tasks for the NSW

The following is a list of not yet completed ELF and ELF related tasks required by SRI-ARC for its NSW work, and our understanding of the current status of these tasks, 1

The ELF KERNEL 1a

We need a TEST and a TESTS (test specific) system call so we can check for the occurence of an event without being put to sleep, 1a1

Status: 1a2

Dave Retz has indicated that it would be trivial to implement these two system calls, but has not yet gotten around to doing it, 1a2a

The ELF EXEC 1b

We need the ELF EXEC in a working and reliable state, 1b1

We need to get a better understanding of the relationships that exist between the ELF KERNEL, the ELF EXEC, and user processes running on ELF, Specifically, it appears that from a users point of view, some system calls are part of the KERNEL and some system calls are part of the EXEC, Since it will eventually be necessary for us to replace the ELF EXEC with an NSW EXEC, we need to know how to separate the ELF EXEC into two parts: 1b2

that part of the EXEC that implements system calls, and 1b2a

that part of the EXEC that serves as the ELF command interpreter, 1b2b

Status: 1b3

The ELF EXEC is supposed to be fully operational by Dec, 1, and documentation on its structure has been promised, but no date set for the documentation, 1b3a

ELF Network Programs 1c

We need a working NCP in ELF, 1c1

We need a working TELNET in ELF, 1c2

Status: 1c3

The ELF NCP and TELNET programs are supposed to be fully operational by Dec. 1. 1c3a

ELF Virtual Memory 1d

We need the virtual memory implementation of ELF. Without this capability, only 28K of the memory on an 11 is usable. 1d1

Status: 1d2

The virtual memory features of ELF are not expected to be ready until at least Jan. 1, 1975. 1d2a

Loading ELF 1e

We need to be able to "boot load" ELF into an 11 from over the network. 1e1

Status: 1e2

Eric Mader of BBN is currently working on this procedure. However, his boot loading procedures appear to require the use of experimental NCP programs. I am not sure of the current state of his work with regards to completion of this task. 1e2a

Loading User Programs 1f

We need to be able to load user processes from over the network. There appear to be several ways to do this: 1f1

1) Have a user FTP that runs on ELF that can get a remote file and store it in core (by using the Inter Process Port capabilites of ELF) rather than on a disk. This seems to be the most desirable approach. 1f1a

2) Have a server FTP that runs on ELF that can receive a remote file and store it in core (by using the Inter Process Port capabilites of ELF) rather than on a disk. In this case we would TELNET to the remote host that holds the file we wish to load and then use FTP on the remote host to send the file to ELF. 1f1b

3) Have a dedicated ELF process (a process that is part of the ELF operating system) that is always listening on a specific socket for files sent to it from a remote host. This process would then store the received file in core. This seems to be the least desirable approach in that it requires initiating action on a remote host and that the

functions performed by this process are so similar to those that would be performed by a user FTP that it seems senseless to have a special separate process. 1f1c

All of these methods seem to require the pre=existance of a process that is waiting to load, via an IPP, the remote file. It would be desirable to have a (load) system call that would set up this process with the approriate address space and IPPs. The FTP server or user process could then issue this sytem call at the right time. 1f2

Status: 1f3

Full server and user FTP processes are planned for ELF, but will probably not be fully operational until Spring, 1975. It appears that we will have to write our own code for the process that will load remote files into core via IPPs. 1f3a

ELF Debugging 1g

We need the ELF debugging process. A debugging process, which has the ability to monitor other processes, has been designed for the ELF operating system. Our debugging plans call for the use of this process. 1g1

Status: 1g2

Eric Mader of BBN is writing and implementing the ELF debugging process. He thinks he will be finished around mid December, 1975. 1g2a

Space Allocation 1h

Given the memory limitations of an 11, it might be nice to have system buffer pool calls. 1h1

Status: 1h2

ADR agreed at the recent NSW meeting to investigate this path. 1h2a

PCP 1i

We need the PCP routines for the implementation of the NSW. 1i1

Status: 1i2

SRI=ARC has most of the design work done and will be starting implementation soon. 1i2a

Documentation 1j

There is a need for more documentation about ELF from both a user's point of view, and from a system programmer's point of view. 1j1

Status: 1j2

Dave Retz has plans for eventually getting around to doing all the required documentation, however, it appears that as usual in the programming world, documentation will not be available until after many of the programming tasks are completed. 1j2a

General Requirements 1k

In general we need an ELF that is reliable and bug free so we can devote ourselves to NSW task without being sidetracked into debugging of ELF. 1k1

Status: 1k2

It is hard to make any statement about the reliability of a system that is not yet in full operational use. 1k2a

The following is our understanding of which groups have responsibility for the above tasks: 2

SCRL Tasks 2a

The ELF KERNEL 2a1

The ELF EXEC 2a2

The ELF Network Programs 2a3

The ELF Virtual Memory Features 2a4

Documentation 2a5

SRI-ARC Tasks 2b

Loading User Programs Over the Network 2b1

We assume we have responsibiltiy for writing any user code necessary for the loading of user programs; it is not clear who has responsibilty for getting an FTP running or for getting any new system calls needed for the support of loading user programs over the network. 2b1a

PCP 2b2

ADR Tasks 2c

Memory Space Allocation 2c1

Maintainance of ELF after it is developed 2c2

BBN Tasks 2d

Loading ELF over the Network 2d1

The ELF Debugging Process 2d2

Conclusions 3

It appears that the 4 programmers working on ELF are overburdened, and that they are doing the best that is humanly possible. It may be desirable to loan them an ADR person to assist in the current development of ELF. (It's possible that this loaned person could be assigned to assist in getting the needed documentation completed.) 3a

At the recent (NOV. 5-6) NSW meeting ADR indicated that it would like to freeze an NSW version of ELF, possibly as early as next month. By that time, as indicated above, many of the features needed by SRI-ARC for its tasks will not be available and therefore to freeze an NSW version of ELF at this time seems premature. 3b

ELF and ELF related tasks for the NSW

(J24575) 22-NOV-74 16:01;;;; Title: Author(s): Kenneth E. (Ken) Victor/KEV; Distribution: /NPG([INFO-ONLY]) RWW([INFO-ONLY]) ; Sub-Collections: SRI-ARC NPG; Clerk: KEV; Origin: < VICTOR, ELF/UNIX.NLS;1, >, 22-NOV-74 15:55 KEV ;;;;####;

PCP Data Structure Formats
PCPFMT Version 2

22-NOV-74

James E. White
Augmentation Research Center

Stanford Research Institute
Menlo Park, California 94025

PCPFMT specifies the defined data structure encodings for the Procedure Call Protocol (PCP -- 24459,), with which the reader of the present document is assumed familiar.

(J24576) 22-NOV-74 16:07;;;; Title: Author(s): James E. (Jim) White/JEW; Sub-Collections: SRI-ARC; Clerk: JEW; Origin: < WHITE, PCP-PCPFMT.NLS;4, >, 22-NOV-74 11:59 JEW ;;;; ####;

PREFACE 1

The Procedure Call Protocol (PCP) is an inter-process and/or inter-host protocol that permits a collection of processes within one or more ARPANET hosts to communicate at the procedure call level. In effect, it makes the component procedures of remote software systems as accessible to the programmer as those within his own system. PCP specifies both a virtual programming environment (VPE) in which remote procedures may be assumed to operate, as well as the inter-process exchanges that implement it. 1a

The Multi-Process Software System (MPSS) whose construction PCP makes practical and of which the NSW is an example, consists of collections of "procedures" and "data stores" called "packages", in one or more "processes", interconnected in a tree structure by "physical channels". Procedures within a process have free access to the procedures (and data stores) of each process adjacent to it in the tree structure, and may call upon them as if they were local subroutines. Superimposed upon the tree structure is a more general set of interconnections which give non-adjacent processes in the tree the same kind of access to one another. 1b

The MPSS is implemented by: 1c

1) low-level protocols which provide the basic, inter-process communicaton (IPC) facilities by which channels are implemented: an inter-host IPC protocol (PCPHST), an inter-Tenex-fork IPC protocol (PCPFRK), and data structure format specifications for both connection types (PCPFMT). 1c1

2) PCP proper, which largely defines the VPE (especially, the procedure call and return mechanism) and specifies the inter-process control exchanges required to implement it. 1c2

3) a set of system packages, implemented within each process, which augment PCP proper by providing mechanisms by which user procedures can: call remote procedures (implemented by the Procedure Interface Package, PIP), manipulate remote data stores (implemented by the PCP Support Package, PSP), and interconnect processes (implemented by the Process Management Package, PMP). 1c3

4) user packages in each process. 1c4

INTRODUCTION 2

This document defines a set of formats for PCP data structures; each is appropriate for one or more physical channel types. Formats are currently specified for channels on which the following kinds of messages can be transmitted: 2a

1) a stream of characters 2a1

2) a stream of 36-bit binary words 2a2

THE PCPTXT FORMAT 3

Introduction 3a

Data structures may be encoded according to PCPTXT when the physical channel allows messages which are ASCII text streams. 3a1

Data Structure Format 3b

```
datastruc ::= [': key]
(charstr/bitstr/integer/boolean/empty/list)               3b1

   key     ::= datastruc                                  3b1a

   charstr ::= 'C length ', lengthschar                   3b1b

      length ::= sdigit                                   3b1b1

         digit ::= one of the digits 0-9                  3b1b1a

      char   ::= one of the 128 ASCII characters          3b1b2

   bitstr  ::= 'B length ', length$('0 / '1)              3b1c

   integer ::= 'I ['-] $digit                             3b1d

   boolean ::= 'T / 'F                                    3b1e

   empty   ::= 'E                                         3b1f

   list    ::= 'L length ', length$datastruc              3b1g
```

THE PCPB36 FORMAT 4

Introduction 4a

Data structures may be encoded according to PCPB36 when the physical channel allows messages which are streams of 36-bit binary words. 4a1

Data Structure Encoding 4b

```
Header (1 word)                                                   4b1
   Bits   0-3 Data type                                           4b1a
      CHARSTR=0  BOOLEAN=3                                        4b1a1
      BITSTR =1  EMPTY  =4                                        4b1a2
      INTEGER=2  LIST   =5                                        4b1a3
   Bits   4-5 Value encoding                                      4b1b
      CHARSTR                                                     4b1b1
         HEADER=0  Value field:                                   4b1b1a
            Character count 'n' (1 word)                          4b1b1a1
            ASCII string ((n+4)/5 words)                          4b1b1a2
         ASCIZ =1  Value field:  ASCIZ string                     4b1b1b
         SIXBIT=2  Value field: SIXBIT string (1 word)            4b1b1c
      BITSTR                                                      4b1b2
         HEADER=0  Value field:                                   4b1b2a
            Bit count 'n'        (1 word)                         4b1b2a1
            Bit string ((n+35)/36 words)                          4b1b2a2
      INTEGER                                                     4b1b3
         TWOSCOMPL=0                                              4b1b3a
            Value field:  Two's complement integer (1 word)       4b1b3a1
      BOOLEAN                                                     4b1b4
         FALSE=0  (Value                                          4b1b4a
         TRUE =1   field                                          4b1b4b
      EMPTY         not                                           4b1b5
         NOTUSED=0   used)                                        4b1b5a
      LIST                                                        4b1b6
         SPECIFIEDELEMENTS=0  Value field:                        4b1b6a
            Element count 'n' (1 word)                            4b1b6a1
            Elements                                              4b1b6a2
         REPEATEDELEMENT=1    Value field:                        4b1b6b
            Element count 'n' (1 word)                            4b1b6b1
            Element to be repeated                                4b1b6b2
         REPEATEDHEADER=2     Value field:                        4b1b6c
            Element count 'n' (1 word)                            4b1b6c1
            Common Header     (1 word)                            4b1b6c2
            Element values                                        4b1b6c3
   Bits  6-13 Unused (zero)                                       4b1c
   Bits 14-17 Gross   key length 'GKL' in words or zero           4b1d
   Bits 18-35 Gross value length 'GVL' in words or zero           4b1e
```

Key (GKL words) 4b2
Value (GVL words) 4b3

PREFACE 1

The Procedure Call Protocol (PCP) is an inter-process and/or inter-host protocol that permits a collection of processes within one or more ARPANET hosts to communicate at the procedure call level. In effect, it makes the component procedures of remote software systems as accessible to the programmer as those within his own system. PCP specifies both a virtual programming environment (VPE) in which remote procedures may be assumed to operate, as well as the inter-process exchanges that implement it. 1a

The Multi-Process Software System (MPSS) whose construction PCP makes practical and of which the NSW is an example, consists of collections of "procedures" and "data stores" called "packages", in one or more "processes", interconnected in a tree structure by "physical channels". Procedures within a process have free access to the procedures (and data stores) of each process adjacent to it in the tree structure, and may call upon them as if they were local subroutines. Superimposed upon the tree structure is a more general set of interconnections which give non-adjacent processes in the tree the same kind of access to one another. 1b

The MPSS is implemented by: 1c

1) low-level protocols which provide the basic, inter-process communicaton (IPC) facilities by which channels are implemented: an inter-host IPC protocol (PCPHST), an inter-Tenex-fork IPC protocol (PCPFRK), and data structure format specifications for both connection types (PCPFMT). 1c1

2) PCP proper, which largely defines the VPE (especially, the procedure call and return mechanism) and specifies the inter-process control exchanges required to implement it. 1c2

3) a set of system packages, implemented within each process, which augment PCP proper by providing mechanisms by which user procedures can: call remote procedures (implemented by the Procedure Interface Package, PIP), manipulate remote data stores (implemented by the PCP Support Package, PSP), and interconnect processes (implemented by the Process Management Package, PMP). 1c3

4) user packages in each process. 1c4

INTRODUCTION 2

This document defines an implementation, appropriate for mediating communication between processes on different hosts within the ARPANET, of the IPC primitives required by PCP. PCPHST uses the PCPB36 data structure format whenever both hosts are Tenex systems, and the PCPTXT format otherwise. Both formats are described in other documents. 2a

The Connection Protocol 2b

ARPANET processes on different hosts communicate by means of a full-duplex Network connection on which both PCP and special "IPC messages" are transmitted. 2b1

PCPHST ports are specified by the following: 2b1a

PORT* ==> %receive socket number% INTEGER 2b1a1

with the corresponding send socket understood to be numbered one greater than the specified receive socket. 2b1b

A process transmits a PCP message by outputting on the connection a special "prefix" followed by the message itself, and then transmitting a Network interrupt (INS) to the remote process via the local NCP. Upon receiving the interrupt, the other process immediately extracts the message from the connection. 2b2

IPC Messages 2c

In addition to sending and receiving PCP messages, the connected processes exchange via the connections, special IPC messages which help to implement the IPC primitives required by PCP. The mechanisms for sending PCP and IPC messages are identical, except for the prefix which preceeds the message on the connection: 2c1

For PCPB36		For PCPTXT:		2c1a
[PCP=0]	(1 word)	↑P	(1 character)	2c1b
[IPC=1]	(1 word)	↑I	(1 character)	2c1c

The currently-defined IPC messages are described in another section of this document. 2c2

IPC IMPLEMENTATION 3

Create process 3a

CRTPRC (prcaddr -> poh, prcname) 3a1

This procedure allocates a table entry indexed by POH, infers an ARPANET host address and contact socket number from PRCADDR, and establishes a full-duplex Network connection with the remote process via the ARPANET Initial Connection Protocol (ICP). The remote process initializes itself and then returns an INITACK IPC message to its superior, specifying its generic process name, which the procedure returns to its caller. 3a2

Each of the two simplex connections which result from the ICP (connection handles to which are stored in the table entry) will be a 36-bit connection governed by the PCPB36 format, if both the local and remote hosts are Tenex systems; otherwise, each will be an 8-bit connection governed by PCPTXT. 3a3

Delete process 3b

DELPRC (poh) 3b1

This procedure outputs a TERM IPC message using one of the connection handles stored in the table entry indexed by POH. The inferior cleans up, returns a TERMACK message to its superior which specifies the cost in cents of the process' execution, and closes the Network connections from its end. The local process deletes them from his end and deletes the table entry. 3b2

Send message to process 3c

SNDMSG (poh, message) 3c1

This procedure outputs the PCP message MESSAGE using one of the connection handles stored in the table entry indexed by POH. 3c2

Accept message from process 3d

RCVMSG (poh -> message) 3d1

This procedure awaits and then inputs the next PCP message MESSAGE using one of the connection handles stored in the table entry indexed by POH, and returns it to the caller. 3d2

Create end of inter-process channel 3e

CRTCHNEND (poh, remport) 3e1

This procedure issues in parallel via its NCP and waits for acknowledgment of, a matched pair of Requests for Connection (RFCs) specifying the local socket pair and remote host saved by ALOPOR in the table entry indexed by POH, and the remote socket pair specified by REMPORT. Once the connections have been established, the procedure saves their handles in the table entry. 3e2

Delete end of inter-process channel 3f

DELCHNEND (poh) 3f1

This procedure closes from its end, the Network connections whose handles are stored in the table entry indexed by POH. 3f2

Allocate local port 3g

ALOPOR (chntypmnu, remloc -> chntypsel, port, poh) 3g1

If both the local host and the host specified by REMLOC are Tenex systems, this procedure selects from CHNTYPMNU the INTERHOST channel type with a width of 36 (bits), if it is offered. Otherwise, it selects the INTERHOST channel type with a width of 8. In either case, it saves the selection for return to the caller as CHNTYPSEL. 3g2

The procedure then saves the remote process' host address and the numbers of a send-receive socket pair which the local process allocates, in a table entry indexed by POH. It then returns the receive socket number to the caller as PORT. 3g3

Release local port 3h

RELPOR (poh) 3h1

This procedure releases the send-receive socket pair associated with the table entry indexed by POH, and the table entry itself. 3h2

IPC MESSAGES 4

Acknowledge initialization of inferior process 4a

INITACK (prcname) 4a1

This message, sent only from inferior to superior, acknowledges the former's initialization and returns the generic process name PRCNAME of the inferior process. 4a2

Format: 4a3

LIST (%opcode% INTEGER [INITACK=0], %prcname% CHARSTR) 4a3a

NOTE: In this and all subsequent descriptions of IPC message formats, only the PCPTXT format (as implied by the PCP data structure) is given. The format which applies when the connection is governed by the PCPB36 format is the same as specified in the PCPFRK document. 4a3b

Terminate 4b

TERM () 4b1

This message, sent only from superior to inferior, requests the latter to terminate execution and respond with a TERMACK message. 4b2

Format: 4b3

LIST (%opcode% INTEGER [TERM=1]) 4b3a

Acnowledge termination of inferior fork 4c

TERMACK (cost) 4c1

This message, sent only from inferior to superior, acknowledges the termination of the former and returns the cost of its use in cents. 4c2

Format: 4c3

LIST (%opcode% INTEGER [TERMACK=2], %cost% INTEGER) 4c3a

Note protocol violation 4d

IPCERR (errcode, errmsg) 4d1

This message notifies the receiving process that the sending process has witnessed it violate the IPC protocol. ERRCODE and ERRMSG (which is optional) identify the error in program- and human-readable form, respectively. 4d2

The superior process (if any) should at least log the error report, and probably break off communication with the inferior. 4d3

Format: 4d4

```
LIST (%opcode% INTEGER [ICPERR=3], %errcode% INTEGER,
      %errmsg% CHARSTR / EMPTY)                          4d4a
```

No operation 4e

NOP () 4e1

This message requests no operation and may be discarded without action by the receiving process. 4e2

Format: 4e3

```
LIST (%opcode% INTEGER [NOP=4])                          4e3a
```

PCP ARPANET Inter-Host IPC Implementation
PCPHST Version 2

22-NOV-74

James E. White
Augmentation Research Center

Stanford Research Institute
Menlo Park, California 94025

PCPHST is the implementation, for ARPANET inter-host communication, of the IPC primitives required by the Procedure Call Protocol (PCP -- 24459,), with which the reader of the present document is assumed familiar.

(J24577) 22-NOV-74 16:09;;;; Title: Author(s): James E. (Jim) White/JEW; Sub-Collections: SRI-ARC; Clerk: JEW; Origin: < WHITE, PCP-PCPHST.NLS;2, >, 22-NOV-74 12:15 JEW ;;;; ####;

PREFACE 1

The Procedure Call Protocol (PCP) is an inter-process and/or inter-host protocol that permits a collection of processes within one or more ARPANET hosts to communicate at the procedure call level. In effect, it makes the component procedures of remote software systems as accessible to the programmer as those within his own system. PCP specifies both a virtual programming environment (VPE) in which remote procedures may be assumed to operate, as well as the inter-process exchanges that implement it. 1a

The Multi-Process Software System (MPSS) whose construction PCP makes practical and of which the NSW is an example, consists of collections of "procedures" and "data stores" called "packages", in one or more "processes", interconnected in a tree structure by "physical channels". Procedures within a process have free access to the procedures (and data stores) of each process adjacent to it in the tree structure, and may call upon them as if they were local subroutines. Superimposed upon the tree structure is a more general set of interconnections which give non-adjacent processes in the tree the same kind of access to one another. 1b

The MPSS is implemented by: 1c

1) low-level protocols which provide the basic, inter-process communicaton (IPC) facilities by which channels are implemented: an inter-host IPC protocol (PCPHST), an inter-Tenex-fork IPC protocol (PCPFRK), and data structure format specifications for both connection types (PCPFMT). 1c1

2) PCP proper, which largely defines the VPE (especially, the procedure call and return mechanism) and specifies the inter-process control exchanges required to implement it. 1c2

3) a set of system packages, implemented within each process, which augment PCP proper by providing mechanisms by which user procedures can: call remote procedures (implemented by the Procedure Interface Package, PIP), manipulate remote data stores (implemented by the PCP Support Package, PSP), and interconnect processes (implemented by the Process Management Package, PMP). 1c3

4) user packages in each process. 1c4

INTRODUCTION 2

This document defines an implementation, appropriate for mediating communication between Tenex forks, of the IPC primitives required by PCP. PCPFRK uses the PCPB36 data structure format, described in another document. 2a

NOTE: 2a1

1) This implementation currently deals only with forks within the same job. Direct PCP channels between forks in different jobs within a single Tenex are therefore not currently supported. 2a1a

2) The implementation of the CRTCHNEND primitive described here is predicated upon the implementation of job-global fork handles in Tenex; PRCNO in PRCLOC* IS such a fork handle. In the absence of that monitor change, PCP will not support direct channels between forks within the same job (except, of course, between a fork and its direct inferiors). 2a1b

The Inter-Fork Window 2b

Connected forks communicate by means of shared pages in their respective address spaces, the intersection of which constitutes a "window" through which both PCP and special "IPC messages" are transmitted. The window has the following format: 2b1

```
LOCK        (1 word)  Window lock                                 2b1a
   AVAILABLE =-1  Window is free                                  2b1a1
   LOCKED    = 0  Window is locked                                2b1a2
   ENQUEUED  > 0  Window is locked and sought by other fork       2b1a3
EOM         (1 bit)   End of message                              2b1b
TYPE       (17 bits)  Message type                                2b1c
   PCP=0                                                          2b1c1
   IPC=1                                                          2b1c2
LENGTH     (18 bits)  Length of MESSAGE in words                  2b1d
MESSAGE (remainder)   Message                                     2b1e
```

The Window Protocol 2c

The window is a half-duplex communication device whose use is controlled by means of the lock LOCK, and an interrupt channel in each of the connected forks' PSI systems. 2c1

PCPFRK ports are specified by the following: 2c1a

```
PORT* ==> LIST (%page% INTEGER, %channel% INTEGER)    2c1a1
```

where PAGE in the page number of the first page in the fork's address space which is part of the window, and CHANNEL is the number of the PSI channel which the fork has allocated to the window. 2c1b

A fork transmits a PCP message through the window by "locking" the window; placing the message and its length in words in the window's MESSAGE and LENGTH fields, respectively; setting the TYPE field to PCP; raising the EOM bit; and interrupting the other fork. The fork locks the window, either by adding one to LOCK and finding the result LOCKED, or by receiving a message from the other fork. 2c2

Upon receiving the interrupt which signals the presence of a message in the window, the other fork immediately removes the message and "unlocks" the window. The fork unlocks the window either by returning a message to the other fork, or by exchanging the contents of LOCK for the value AVAILABLE and, if LOCK is discovered to have been ENQUEUED, resetting LOCK to LOCKED and sending a NOP IPC message (described below) to the other fork. 2c3

Both forks agree to promptly unlock the window after each message, in most cases even before the message is processed, leaving the window available to either fork for transmission of another messages. 2c4

IPC Messages 2d

In addition to sending and receiving PCP messages, the connected forks exchange via the window, special IPC messages which help to implement the IPC primitives required by PCP. The mechanisms for sending PCP and IPC messages are identical, except that the sender stores IPC, rather than PCP, in the window's TYPE field. The currently-defined IPC messages are described in another section of this document. 2d1

Multi-Packet Messages 2e

Since the window is of finite size, some messages may overflow the MESSAGE field. In such cases, the sender is permitted to transmit the message in two or more "packets". The mechanisms for sending a whole message and a packet of a message are identical, except that in the latter case, the EOM bit is raised only on the last packet, and the sender maintains control of the window until that last packet has been sent. The receiving IPC code must concatenate the packets to reconstruct the full message. 2e1

IPC IMPLEMENTATION 3

Create process 3a

CRTPRC (prcaddr => poh, prcname) 3a1

This procedure allocates a table entry indexed by POH, infers a SAV file name from PRCADDR, creates an inferior fork whose handle it stores in the table entry, maps the file into the inferior fork, stores the following parameters in the fork's ACs: 3a2

0 Superior's proposed window XWD SL,SU 3a2a
1 Superior's interrupt channel number 3a2b

and dispatches it at its entry point. SL and SU are page numbers which define the segment of its address space which the superior is prepared to devote to the window. The inferior initializes itself and then returns via HALTF to its superior, who extracts the following from the inferior's ACs: 3a3

0 Inferior's proposed window XWD IL,IU 3a3a
1 Inferior's interrupt channel number 3a3b

The procedure then establishes via the appropriate map operations, the following compromise windows in the inferior's and superior's address spaces, respectively: 3a4

XWD IL, IL + MINIMUM (IU-IL, SU-SL) 3a4a
XWD SL, SL + MINIMUM (IU-IL, SU-SL) 3a4b

stores the parameters of the latter in the table entry, and restarts the inferior. At this point, initialization of the window is complete. The inferior sends an INITACK IPC message to the superior, specifying its generic process name, which the procedure returns to its caller. 3a5

Delete process 3b

DELPRC (poh) 3b1

This procedure transmits a TERM IPC message to the inferior fork whose handle is stored in the table entry indexed by POH. The inferior cleans up, returns a TERMACK message to its superior which specifies the cost in cents of the process' execution, and halts via HALTF. The local fork then deletes the window, via the appropriate map operations; the fork itself, via KFORK; and the table entry. 3b2

Send message to process 3c

SNDMSG (poh, message) 3c1

This procedure transmits the PCP message MESSAGE to the fork whose handle is stored in the table entry indexed by POH. 3c2

Accept message from process 3d

RCVMSG (poh => message) 3d1

This procedure awaits and then accepts the next PCP message MESSAGE from the fork whose handle is stored in the table entry indexed by POH, and returns it to the caller. 3d2

Create end of inter-process channel 3e

CRTCHNEND (poh, remport) 3e1

This procedure is a NOP if the remote fork's handle (saved by ALOPOR in the table entry indexed by POH) is smaller than the local fork's. Otherwise, the procedure creates the window arranged by ALOPOR (whose parameters are also stored in the table entry), using the appropriate map operations. 3e2

Delete end of inter-process channel 3f

DELCHNEND (poh) 3f1

This procedure is a NOP if the remote fork's handle (saved by ALOPOR in the table entry indexed by POH) is smaller than the local fork's. Otherwise, the procedure deletes the window arranged by ALOPOR (whose parameters are stored also in the table entry), using the appropriate map operations. 3f2

Allocate local port 3g

ALOPOR (chntypmnu, remloc -> chntypsel, port, poh) 3g1

This procedure tentatively allocates for an IPC window, a segment of the local fork's address space whose width is probably a local constant. It then selects from CHNTYPMNU the INTERPRC channel type which maximizes the mininum of the tentative window width and the window width offered in the selection. Using the compromise channel width, the procedure constructs a CHNTYPSEL for return to the caller. 3g2

The procedure then firmly allocates a window of the compromise width and returns as PORT, the number of the first page in the window and the number of a local PSI channel it allocates. In a table entry indexed by POH, the procedure saves the window parameters and the other fork's handle which it extracts from REMLOC (whose HOSTADDR and JOBNO fields are known to match those of the local fork). 3g3

Release local port 3h

RELPOR (poh) 3h1

This procedure releases the window and PSI channel associated with the table entry indexed by POH, and the table entry itself. 3h2

IPC MESSAGES 4

Acknowledge initialization of inferior fork 4a

INITACK (prcname) 4a1

This message, sent only from inferior to superior, acknowledges the former's initialization and returns the generic process name PRCNAME of the inferior process. 4a2

Format: 4a3

```
opcode [INITACK=0]  (1 word)                                4a3a
prcname         (ASCIZ string)                              4a3b
```

Terminate 4b

TERM () 4b1

This message, sent only from superior to inferior, requests the latter to terminate execution and respond with a TERMACK message. 4b2

Format: 4b3

```
opcode [TERM=1]  (1 word)                                   4b3a
```

Acnowledge termination of inferior fork 4c

TERMACK (cost) 4c1

This message, sent only from inferior to superior, acknowledges the termination of the former and returns the cost of its use in cents. 4c2

Format: 4c3

```
opcode [TERMACK=2]  (1 word)                                4c3a
cost                (1 word)                                4c3b
```

Note protocol violation 4d

IPCERR (errcode, errmsg) 4d1

This mssage notifies the receiving fork that the sending fork has witnessed it violate the window protocol. ERRCODE and ERRMSG (which is optional, i.e. may be null) identify the error in program- and human-readable form, respectively. 4d2

The superior fork (if any) should at least log the error report, and probably break off communication with the inferior. 4d3

Format: 4d4

```
opcode [ICPERR=3]   (1 word)          4d4a
errcode             (1 word)          4d4b
errmsg        (ASCIZ string)          4d4c
```

No operation 4e

NOP () 4e1

This message requests no operation and may be discarded without action by the receiving fork. It is used primarily, as described earlier, to unlock the window. 4e2

Format: 4e3

```
opcode [NOP=4]  (1 word)              4e3a
```

PCP Tenex Inter-Fork IPC Implementation
PCPFRK Version 2

22-NOV-74

James E. White
Augmentation Research Center

Stanford Research Institute
Menlo Park, California 94025

PCPFRK is the implementation, for Tenex inter-fork communication, of the IPC primitives required by the Procedure Call Protocol (PCP -- 24459,), with which the reader of the present document is assumed familiar.

(J24578) 22-NOV-74 16:12;;;; Title: Author(s): James E. (Jim) White/JEW; Sub-Collections: SRI-ARC; Clerk: JEW; Origin: < WHITE, PCP-PCPFRK.NLS;8, >, 22-NOV-74 12:11 JEW ;;;; ####;

The Low-Level Debug Package
LLDBUG Version 2

22-NOV-74

James E. White
Augmentation Research Center

Stanford Research Institute
Menlo Park, California 94025

LLDBUG is a debugging tool that operates within the setting provided by the Procedure Call Protocol (PCP -- 24459,), with which the reader of the present document is assumed familiar.

(J24579) 22-NOV-74 16:18;;;; Title: Author(s): James E. (Jim) White/JEW; Sub-Collections: SRI-ARC; Clerk: JEW; Origin: < WHITE, PCP-LLDBUG.NLS;6, >, 22-NOV-74 13:24 JEW ;;;; ####;

PREFACE 1

The Procedure Call Protocol (PCP) is an inter-process and/or inter-host protocol that permits a collection of processes within one or more ARPANET hosts to communicate at the procedure call level. In effect, it makes the component procedures of remote software systems as accessible to the programmer as those within his own system. PCP specifies both a virtual programming environment (VPE) in which remote procedures may be assumed to operate, as well as the inter-process exchanges that implement it. 1a

The Multi-Process Software System (MPSS) whose construction PCP makes practical and of which the NSW is an example, consists of collections of "procedures" and "data stores" called "packages", in one or more "processes", interconnected in a tree structure by "physical channels". Procedures within a process have free access to the procedures (and data stores) of each process adjacent to it in the tree structure, and may call upon them as if they were local subroutines. Superimposed upon the tree structure is a more general set of interconnections which give non-adjacent processes in the tree the same kind of access to one another. 1b

The MPSS is implemented by: 1c

1) low-level protocols which provide the basic, inter-process communicaton (IPC) facilities by which channels are implemented: an inter-host IPC protocol (PCPHST), an inter-Tenex-fork IPC protocol (PCPFRK), and data structure format specifications for both connection types (PCPFMT). 1c1

2) PCP proper, which largely defines the VPE (especially, the procedure call and return mechanism) and specifies the inter-process control exchanges required to implement it. 1c2

3) a set of system packages, implemented within each process, which augment PCP proper by providing mechanisms by which user procedures can: call remote procedures (implemented by the Procedure Interface Package, PIP), manipulate remote data stores (implemented by the PCP Support Package, PSP), and interconnect processes (implemented by the Process Management Package, PMP). 1c3

4) user packages in each process. 1c4

INTRODUCTION 2

The Low-Level Debug Package (package name=LLDBUG) contains those procedures and data stores which a remote process requires to debug at the assembly-language level, any process known to the local process. The package contains procedures for manipulating and searching the process' address space, for manipulating and searching its symbol tables, and for setting and removing breakpoints from its address space,. Its data stores hold process characteristics and state information, and the contents of program symbol tables. 2a

The procedures in this package are appropriately applied to any process whose processors can each be usefully modelled as shared code and private data in a single address space. 2b

Throughout this document, the following shorthands denote, respectively, a program symbol, and an address in either absolute or symbolic form: 2c

```
SYMBOL*  ==> <tblname> %symname% CHARSTR                      2c1
ADDRESS* ==> INTEGER / LIST (SYMBOL*, %offset% INTEGER)       2c2
```

Recommended Process Development Strategy 2d

Each LLDBUG procedure manipulates a process known to the local process via a handle specified as an argument to the procedure. The local process can therefore be requested, via its OPNPRC procedure, to debug any process known to it (including itself, its superior, a direct inferor, and processes which the invoking process might make known to it via PMP's ITDPRCS procedure). 2d1

In practice, however, the local process is probably capable of debug-level manipulation of only a subset of those processes. In particular, its operating system may permit it to exert such control only over inferior processes. A recommended development strategy, therefore, is to run processes, at least during the checkout stage, as a direct inferior of a special debug process, provided specifically for that purpose. 2d2

PROCEDURES 3

Debug preparations 3a

Open process for debugging 3a1

OPNPRC (ph) 3a1a

This procedure opens for debugging, the process known to the local process via PH. 3a1b

Argument/result types: 3a1c

ph- INTEGER 3a1c1

Close process after debugging 3a2

CLSPRC (ph) 3a2a

This procedure closes after debugging, the process known to the local process via PH. 3a2b

Argument/result types: 3a2c

ph- INTEGER 3a2c1

The Address Space 3b

Read address space 3b1

RDCORE (ph, strtaddr, wrdcnt, encoding => values) 3b1a

This procedure retrieves from the address space of the process known to the local process via PH, the current contents VALUES of the contiguous block of WRDCNT words beginning at address STRTADDR. ENCODING specifies the manner in which the contents of each word are to be encoded for return: 3b1b

```
TEXT:    as text (result type = CHARSTR)                       3b1b1
CODE:    as an executable instruction (result type =
         CHARSTR)                                              3b1b2
INTEGER: as a signed integer (result type = INTEGER)           3b1b3
WORD:    uninterpreted (result type = BITSTR)                  3b1b4
```

Argument/result types: 3b1c

```
ph       = INTEGER                                                3b1c1
strtaddr= ADDRESS*                                                3b1c2
wrdcnt   = INTEGER                                                3b1c3
encoding= INTEGER [TEXT=0 / CODE=1 / INTEGER=2 / WORD=3]          3b1c4
values   = LIST (CHARSTR / INTEGER / BITSTR, ...)                 3b1c5
```

Write address space 3b2

WRCORE (ph, strtaddr, wrdcnt, values, encoding) 3b2a

This procedure replaces the current contents of the contiguous block of WRDCNT words beginning at address STRTADDR in the address space of the process known to the local process via PH, with the new values VALUES. ENCODING specifies the manner in which the new contents of each word have been encoded by the invoking process (same as in RDCORE). 3b2b

Argument/result types: 3b2c

```
ph       = INTEGER                                                3b2c1
strtaddr= ADDRESS*                                                3b2c2
wrdcnt   = INTEGER                                                3b2c3
values   = LIST (CHARSTR / INTEGER / BITSTR, ...)                 3b2c4
encoding= INTEGER [TEXT=0 / CODE=1 / INTEGER=2 / WORD=3]          3b2c5
```

Search address space 3b3

SEARCH (ph, strtaddr, wrdcnt, value, encoding, mask -> addrs) 3b3a

This procedure searches the contiguous block of WRDCNT words beginning at address STRTADDR in the address space of the process known to the local process via PH, for those words ADDRS whose content matches VALUE, after both have been ANDed with the mask MASK. ENCODING specifies the manner in which the comparand VALUE has been encoded by the invoking process (same as in WRCORE). 3b3b

Argument/result types: 3b3c

```
ph       - INTEGER                                              3b3c1
strtaddr- ADDRESS*                                              3b3c2
wrdcnt   - INTEGER                                              3b3c3
value    - CHARSTR / INTEGER / BITSTR                           3b3c4
encoding- INTEGER [TEXT=0 / CODE=1 / INTEGER=2 / WORD=3]        3b3c5
mask     - BITSTR                                               3b3c6
addrs    - LIST (ADDRESS*, ...)                                 3b3c7
```

Symbol tables 3c

Open symbol table 3c1

OPSYMT (Ph, tblname) 3c1a

This procedure opens the symbol table TBLNAME for the process known to the local process via PH. 3c1b

Argument/result types: 3c1c

```
ph      - INTEGER      3c1c1
tblname- CHARSTR       3c1c2
```

Close symbol table 3c2

CLSYMT (ph, tblname) 3c2a

This procedure closes the previously-opened symbol table TBLNAME for the process known to the local process via PH. 3c2b

Argument/result types: 3c2c

```
ph      - INTEGER      3c2c1
tblname- CHARSTR       3c2c2
```

Create symbol 3c3

CRTSYM (Ph, symbol, value) 3c3a

This procedure adds the symbol SYMBOL with value VALUE to one of the previously-opened symbol tables (implicitly named by SYMBOL) for the process known to the local process via PH. 3c3b

Argument/result types: 3c3c

```
ph     - INTEGER       3c3c1
symbol- SYMBOL*        3c3c2
value - ADDRESS*       3c3c3
```

Delete symbol 3c4

DELSYM (ph, symbol) 3c4a

This procedure deletes the symbol SYMBOL from one of the previously-opened symbol tables (implicitly named by SYMBOL) for the process known to the local process via PH. 3c4b

Argument/result types: 3c4c

```
ph     = INTEGER                                   3c4c1
symbol= SYMBOL*                                    3c4c2
```

Read symbol value 3c5

RDSYM (ph, symbol => value) 3c5a

This procedure returns the value VALUE of the symbol SYMBOL in one of the previously-opened symbol tables (implicitly named by SYMBOL) for the process known to the local process via PH. 3c5b

Argument/result types: 3c5c

```
ph     = INTEGER                                   3c5c1
symbol= SYMBOL*                                    3c5c2
value = INTEGER                                    3c5c3
```

Write symbol value 3c6

WRSYM (ph, symbol, value) 3c6a

This procedure assigns the value VALUE to the symbol SYMBOL in one of the previously-opened symbol tables (implicitly named by SYMBOL) for the process known to the local process via PH. 3c6b

Argument/result types: 3c6c

```
ph     = INTEGER                                   3c6c1
symbol= SYMBOL*                                    3c6c2
value = ADDRESS*                                   3c6c3
```

Fit value to symbol table 3c7

FITVAL (ph, comparand, tblname -> symbol, value) 3c7a

This procedure returns the name SYMBOL and value VALUE of the symbol, in the previously-opened symbol table TBLNAME for the process known to the local process via PH (or in any of its symbol tables, if TBLNAME is EMPTY), whose current value is closest to COMPARAND. 3c7b

Argument/result types: 3c7c

ph	= INTEGER	3c7c1
comparand	= ADDRESS*	3c7c2
tblname	= CHARSTR / EMPTY	3c7c3
symbol	= SYMBOL*	3c7c4
value	= INTEGER	3c7c5

Breakpoints 3d

Create breakpoint 3d1

SETBRK (ph, addr, pcdcnt) 3d1a

This procedure sets a breakpoint at address ADDR in the address space of the process known to the local process via PH. The PCDCNTth time the breakpoint is reached by the process, the breakpointed processor's state will be stored in PRCSTA, the primitive: 3d1b

NOTE (BRKPNT, LIST (ph, addr)) 3d1b1

will be invoked (suspending the processor), the processor's state will be restored from PRCSTA, and it will continue execution. 3d1c

The parameters returned by NOTE -- PH and ADDR -- specify, respectively, the handle by which the breakpointed process is known to the local process and the address in its address space at which the breakpoint occurred. 3d1d

Needless to say, the invoking process must lie along the thread of control if it expects to intercept the NOTE. If a second processor within the process encounters a breakpoint, its NOTE will be delayed until the first is complete. 3d1e

Argument/result types: 3d1f

```
ph     = INTEGER       3d1f1
addr   = ADDRESS*      3d1f2
pcdcnt= INTEGER        3d1f3
```

Delete breakpoint 3d2

REMBRK (ph, addr) 3d2a

This procedure removes the breakpoint previously set at address ADDR in the address space of the process known to the local process via PH or, if ADDR is EMPTY, removes all breakpoints from its address space. 3d2b

Argument/result types: 3d2c

```
ph  = INTEGER             3d2c1
addr= ADDRESS* / EMPTY    3d2c2
```

Execute intruction 3d3

EXINST (Ph, inst, encoding) 3d3a

This procedure, callable only while the process known to the local process via PH has a breakpoint NOTE outstanding, restores the breakpointed processor's state from PRCSTA, executes the single instruction INST, and then updates PRCSTA again. ENCODING specifies the manner in which INST has been encoded by the invoking process (same as in WRCORE). 3d3b

Argument/result types: 3d3c

```
ph       - INTEGER                                             3d3c1
inst     - CHARSTR / INTEGER / BITSTR                          3d3c2
encoding- INTEGER [TEXT=0 / CODE=1 / INTEGER=2 / WORD=3]       3d3c3
```

DATA STORES 4

PRCCHR Characteristics of open processes 4a

This read-only data store contains certain characteristic information about each open process. 4a1

PRCCHR is somewhat process-dependent in format and content, but always contains at least the number of words ASIZE in the process's address space, and the width WRDLEN in bits of each word. The MAXLEN of each argument or result of type BITSTR for LLDBUG procedures which apply to that process is given by WRDLEN, as well. 4a2

Data structure type: 4a3

```
<prcchr> LIST (<%ph% INTEGER> LIST (<asize> INTEGER,
         <wrdlen> INTEGER, any, ...), ...)              4a3a
```

PRCSTA States of breakpointed processes 4b

This data store contains the state of the currently breakpointed processor in each open process. 4b1

PRCSTA is somewhat process-dependent in format and content, but always contains at least the contents of the processor's program counter PC and its general registers REGS (if any). 4b2

Data structure type: 4b3

```
<prcsta> LIST (<%ph% INTEGER> LIST (<pc> ADDRESS*, <regs>
         LIST (BITSTR, ...), any, ...) / EMPTY, ...)    4b3a
```

SYMTBS Symbol tables for open processes 4c

This read-only data store contains all of the open symbol tables for each open process, giving the name SYMBOL and value VALUE of each symbol in each open table TBLNAME. 4c1

Data structure type: 4c2

```
<symtbs> LIST (<%ph% INTEGER> LIST (<tblname> LIST (<symbol>
         %value% INTEGER, ...), ...), ...)              4c2a
```

The Executive Package
EXEC Version 2

22-NOV-74

Jon Postel
Augmentation Research Center

Stanford Research Institute
Menlo Park, California 94025

The Executive Package (EXEC) is a set of tool management and measurement procedures that operates within the setting provided by the Procedure Call Protocol (PCP -- 24459,), with which the reader of the present document is assumed familiar.

;

1

PREFACE 2

The Procedure Call Protocol (PCP) is an inter-process and/or inter-host protocol that permits a collection of processes within one or more ARPANET hosts to communicate at the procedure call level. In effect, it makes the component procedures of remote software systems as accessible to the programmer as those within his own system. PCP specifies both a virtual programming environment (VPE) in which remote procedures may be assumed to operate, as well as the inter-process exchanges that implement it. 2a

The Multi-Process Software System (MPSS) whose construction PCP makes practical and of which the NSW is an example, consists of collections of "procedures" and "data stores" called "packages", in one or more "processes", interconnected in a tree structure by "physical channels". Procedures within a process have free access to the procedures (and data stores) of each process adjacent to it in the tree structure, and may call upon them as if they were local subroutines. Superimposed upon the tree structure is a more general set of interconnections which give non-adjacent processes in the tree the same kind of access to one another. 2b

The MPSS is implemented by: 2c

1) low-level protocols which provide the basic, inter-process communicaton (IPC) facilities by which channels are implemented: an inter-host IPC protocol (PCPHST), an inter-Tenex-fork IPC protocol (PCPFRK), and data structure format specifications for both connection types (PCPFMT). 2c1

2) PCP proper, which largely defines the VPE (especially, the procedure call and return mechanism) and specifies the inter-process control exchanges required to implement it. 2c2

3) a set of system packages, implemented within each process, which augment PCP proper by providing mechanisms by which user procedures can: call remote procedures (implemented by the Procedure Interface Package, PIP), manipulate remote data stores (implemented by the PCP Support Package, PSP), and interconnect processes (implemented by the Process Management Package, PMP). 2c3

4) user packages in each process. 2c4

INTRODUCTION 3

The Executive Package (package name = EXEC) contains the procedures and data stores for user identification, accounting, and usage information on the tool bearing host where the Executive Package resides. 3a

PROCEDURES 4

Login process 4a

LOGIN (user, password, account) 4a1

This procedure associates the use of this local process with a USER for access control purposes, protected by the password PASSWORD, and an account ACCOUNT for billing purposes. The arguments USER and ACCOUNT are stored in the data store USERACCT. 4a2

Argument/result types: 4a3

```
user     - CHARSTR     4a3a
password - CHARSTR     4a3b
account  - CHARSTR     4a3c
```

DATA STORES 5

COST Cost of usage 5a

This is a read-only data store which is a list of the accumulated cost in cents by package for the usage since creation of this process. When the cost associated with package handle zero is read the total cost of all packages in the process is reported. 5a1

Data structure type: 5a2

<cost> LIST (<%pkh%INTEGER>%cents%INTEGER, ...) 5a2a

USERACCT User and account currently logged in 5b

This read-only data store contains the name and account of the currently logged in user of this process. The values are set by the LOGIN procedure. 5b1

Data structure type: 5b2

<useracct> LIST (<user>CHARSTR / EMPTY, <account>CHARSTR / EMPTY) 5b2a

USAGE Usage statistics 5c

The current usage statistics of this host system are available in this read-only data store. The usage is characterized by such parameters as number of active users USERS, free core/disk space SPACE, cpu utilization CPU, and scheduled downtime SCHD. 5c1

Data structure type: 5c2

<usage> LIST (%users%INTEGER, %space%INTEGER, %cpu%INTEGER, %schd%CHARSTR) 5c2a

(J24580) 22-NOV-74 16:32;;;; Title: Author(s): Jonathan B. Postel/JBP; Sub-Collections: SRI-ARC; Clerk: JBP; Origin: < POSTEL, NSW-EXEC.NLS;10, >, 22-NOV-74 16:30 JBP ;;;;####;

NOTE: This document is a preliminary suggestion of constraints and policies to be used in the implemmentation of the standard ARPANET host to host protocol for NSW uses. This specification is subject to change as indicated by your commennts. 1

Introduction 2

The NSW higher level protocols assume that the host level protocol will provide reliable transmission of messages which are delivered in order. The host level protocol is assumed to contain flow control mechanisms to prevent the senders of messages from flooding a receiver of messages. The host level protocol is to provide a mechanism for an "out of band" interrupt signal. 2a

The initial implementation of the NSW will use the standard host to host protocol of the ARPA Network. This is the protocol specified in NIC 8246, 2b

Mckenzie, A. "Host/Host Protocol for the ARPA Network," Jan-72. 2b1

There will be some constraints placed on the implementations of this protocol when used in the NSW. The main areas of constraint are the policy used for determining when to send allocate commands, and the policy on waiting for RFNMs. 2c

Allocation and buffer Policy 3

For each NSW receive connection the following allocation policy is used. First define three constants: U, the upper bound; L, the lower bound; and I, the increment. When the connection is first opened the initial allocation is U. 3a

Also define three variables: A, the amount allocated; F, the free space in the buffer; and B, the busy space in the buffer. 3b

Note that the free space is that space which is not committed, the empty space consists of the free space and the allocated but as yet unused space. 3b1

The sum A + F + B will always equal U. 3c

When data arrives allocated space is converted to busy space. When data is consumed busy space is converted to free space. Thus the amount allocated decreases until it reaches the lower bound, L. 3d

At this point an additional allocation message is sent in the amount of the free buffer space, but only if this is at least equal to the minimum increment, I. 3e

The following six quantities are the constants and variables used in making decisions in this allocation policy. 3e1

U = upper bound 3e1a
L = lower bound 3e1b
I = minimum increment 3e1c
A = amount allocated 3e1d
F = amount free 3e1e
B = amount busy 3e1f

The following four formulations describe the relationships between these quantities. 3e2

```
[1] A + F + B = U                                   3e2a

[2] n data characters received                      3e2b
   A <- A - n                                       3e2b1
    B <- B + n                                      3e2b2

[3] n data characrers consumed                      3e2c
   B <- B - n                                       3e2c1
   F <- F + n                                       3e2c2

[4] if A < or = L and F = or > I then               3e2d
   Allocate F                                       3e2d1
   A <- A + F                                       3e2d2
   F <- 0                                           3e2d3
```

The NSW will require that the size of the receive buffer for each connection be at least 8000 bits, and this is therefore the minimum value of U. L shall be one half U, and I shall be one eighth U. 3f

U = 8000 bits. 3f1

L = U/2 3f2

I = U/8 3f3

These values are specified here as an initial selection to test the policy. It is expected that experience will show that perhaps some other values would be better, if and when such a determination is made these values will be respecified. 3f4

Ready for Next Message Policy 4

The host to host protocol specifications require that after sending a message on a connection (link) the sending NCP should wait for a RFNM before sending another message on that connection. 4a

Changes in the treatment of the link number field by the IMP have made possible a different policy. 4b

The link number field has been renamed the message identification field and extended from 8 bits to 12 bits. 4c

If the NCP uses the the additional four bits as a sequence counter it could send several messages before receiving the RFNM for the first message. A four bit cyclic sequence counter would allow up to eight messages to be outstanding at a time. 4d

The NSW hosts should use this policy of multiple outstanding messages on a connection for the connections used in the NSW. 4e

The Leader of a Host to Host message is: 4e1

```
       8         8          12          4
  *---------*---------*--------------*-----*
  !         !         !              !     !
  !  flags  !  host   ! message id   !     !
  !         !         !              !     !
  *---------*---------*--------------*-----*
   0       7 8       1 1            2 2   3
                     5 6            7 8   1
```
4e1a

The Message Id field is broken into the link number and sequence number: 4e2

```
       8       4
  *--------*-----*
  !        !     !
  ! link # !seq#!
  !        !     !
  *--------*-----*
   0      7 8   1
                1
```
4e2a

For each NSW connection the NCP shall send messages using the sequence number part of the identification field on a per connection basis to identify the messages on that link and use the sequence number in the returned RFNM (or Incomplete Transmission) to determine if the message has been delivered and is no longer in the network. 4e3

Retransmission Policy 5

Each message transmitted on an NSW connection should be saved until a RFNM is returned for that message (as determined by the link and sequence numbers). If instead of a RFNM an Incomplete Transmission or Host Dead response is received, then that message should be retransmitted K times. 5a

K is initially set to 10. 5a1

Note that the Allocation policy is a constraint on the receive side of a connection that is completely within in the protocol and that it is a policy that the send side must be prepared to accept. 6

Note also that the RFNM and Retransmission policies are a modification to the send side only and cannot be detected by the receive side. 7

Thus, these policies can be used by NSW host for their interactions with both other NSW hosts and non NSW hosts. 8

NSW Host Protocol
Version 2

22-NOV-74

Jon Postel
Augmentation Research Center

Stanford Research Institute
Menlo Park, California 94025

The National Software Works host level protocol is (in the intitial version) a slightly constrained form of the standard ARPA Network host to host protocol.

(J24581) 22-NOV-74 16;54;;;; Title; Author(s); Jonathan B, Postel/JBP; Sub-Collections; SRI-ARC; Clerk; JBP; Origin; < POSTEL, NSW-HOST.NLS;10, >, 22-NOV-74 13;46 JBP ;;;; ####;

,

1

PREFACE 2

The Procedure Call Protocol (PCP) is an inter-process and/or inter-host protocol that permits a collection of processes within one or more ARPANET hosts to communicate at the procedure call level. In effect, it makes the component procedures of remote software systems as accessible to the programmer as those within his own system. PCP specifies both a virtual programming environment (VPE) in which remote procedures may be assumed to operate, as well as the inter-process exchanges that implement it. 2a

The Multi-Process Software System (MPSS) whose construction PCP makes practical and of which the NSW is an example, consists of collections of "procedures" and "data stores" called "packages", in one or more "processes", interconnected in a tree structure by "physical channels". Procedures within a process have free access to the procedures (and data stores) of each process adjacent to it in the tree structure, and may call upon them as if they were local subroutines. Superimposed upon the tree structure is a more general set of interconnections which give non-adjacent processes in the tree the same kind of access to one another. 2b

The MPSS is implemented by: 2c

1) low-level protocols which provide the basic, inter-process communicaton (IPC) facilities by which channels are implemented: an inter-host IPC protocol (PCPHST), an inter-Tenex-fork IPC protocol (PCPFRK), and data structure format specifications for both connection types (PCPFMT). 2c1

2) PCP proper, which largely defines the VPE (especially, the procedure call and return mechanism) and specifies the inter-process control exchanges required to implement it. 2c2

3) a set of system packages, implemented within each process, which augment PCP proper by providing mechanisms by which user procedures can: call remote procedures (implemented by the Procedure Interface Package, PIP), manipulate remote data stores (implemented by the PCP Support Package, PSP), and interconnect processes (implemented by the Process Management Package, PMP). 2c3

4) user packages in each process. 2c4

INTRODUCTION 3

The File Package (package name = FP) contains those procedures and data stores which a remote process requires to employ the file storage and transfering services of the local process. The package contains procedures for opening, closing, and listing directories, for creating, deleting, and renaming files, and for outputting, updating, and deleting files and elements of files. It also contains data stores of directory and file descriptors. 3a

Files 3b

Introduction 3b1

A "file" is a named PCP data structure, stored not in the local process's address space, but on secondary storage. A file thus has an indefinite lifetime, and in particular is not destroyed by the deletion of its local process. Files are manipulated via procedures provided by the file package, rather than via the PCP Support Package's RDDATA and WRDATA procedures. 3b1a

A file, like any PCP data structure, can be arbitrarily complex. 3b1b

There are a few data structures which will describe a very large fraction of the files in actual use; for these files the following structure types are identified: 3b1c

1) Unstructured binary files 3b1c1

seqbin= BITSTR 3b1c1a

2) Paged (and possibly holey) binary files 3b1c2

paged= LIST (%att%CHARSTR, <%pgno%INTEGER>%page%BITSTR, ...) 3b1c2a

3) Unstructured text files 3b1c3

seqtxt= CHARSTR 3b1c3a

4) Record-structured text files 3b1c4

rectxt= LIST (CHARSTR, ...) 3b1c4a

Associated with a file is a "use type". It is expected that there will be many use types. Some examples of use types are: 3b1d

NLS.SRC 3b1d1

COBOL.SRC 3b1d2

COBOL.LIST 3b1d3

360.REL 3b1d4

TENEX.SAV 3b1d5

TECO.SRC 3b1d6
ANY.PRINT 3b1d7

Use types would be utilized by the works manager when preparing for tool use to check for consistency between the intended use of the file and the input expected by the tool. 3b1e

Use type mismatches could result in a call for a file conversion procedure to be executed, which could create a new file with the proper use type. 3b1f

It is expected that there will be many use types which map into a few structure types. 3b1g

Access Controls 3b2

The "creator" of a file can independently grant or refuse the following types of access to it: 3b2a

1) READ: the right to read the file (with the GETFIL procedure), 3b2a1
2) WRIT: the right to modifiy, delete, or rename the file (with the PUTFIL, DELELM, DELFIL, and RENFIL procedures), and 3b2a2
3) CTRL: the right to modify the access assignments themselves. 3b2a3

to the following classes of users: 3b2b

1) CRT: the creator himself, 3b2b1
2) MEM: a directory member, i.e. anyone with its password (described more fully later), and 3b2b2
3) PUB: the general public. 3b2b3

The access assignments for the file are stored in the file's "access descriptor": 3b2c

```
ACCDSC* ==> LIST (crt, mem, pub)                              3b2c1

crt- LIST (%read% BOOLEAN, %writ% BOOLEAN, %ctrl%
     BOOLEAN)                                                 3b2c2
mem- LIST (%read% BOOLEAN, %writ% BOOLEAN, %ctrl%
     BOOLEAN)                                                 3b2c3
pub- LIST (%read% BOOLEAN, %writ% BOOLEAN, %ctrl%
     BOOLEAN)                                                 3b2c4
```

The access descriptor is specified initially when the file is created (via the CRTFIL procedure), and can be modified any time thereafter by anyone with controlling access to the file. 3b2d

File Descriptors 3b3

Associated with every file is a secondary data structure called a "file descriptor", which contains information about the file and which has the following format: 3b3a

```
FILDSC* ==> LIST (                                              3b3a1

   <use>     %use type% CHARSTR,                                3b3a1a
   <crtor>   %file creator% CHARSTR,                            3b3a1b
   <accdsc>  %access descriptor% ACCDSC*,                       3b3a1c
   <crdat>   %creation date and time% CHARSTR,                  3b3a1d
   <rddat>   %date and time of last read% CHARSTR,              3b3a1e
   <wrdat>   %date and time of last write% CHARSTR,             3b3a1f
   <acct>    %account% CHARSTR)                                 3b3a1g
```

Directories 3c

Introduction 3c1

The files within a process are partitioned into one or more "directories", Directories are referred to initially (in the OPNDIR procedure) by name, and thereafter via a "directory identifier", or DID, A directory is "known" if and only if it has been successfully "opened" (i,e, if a DID has been obtained for it), 3c1a

NOTE: the "LOGIN directory" (if any) implied by the USERname last specified via EXEC's LOGIN procedure is always considered open (with DID=0) and need not, indeed cannot, be explicitly opened or closed (with OPNDIR and CLSDIR), 3c1a1

Access Controls 3c2

The "creator" of a directory can independently grant or refuse the following types of access to it: 3c2a

1) READ: the right to open and list the directory (with the OPNDIR and LSTDIR procedures), 3c2a1
2) WRIT: the right to change the file names in the directory (with the CRTFIL, DELFIL, or RENFIL procedures), and 3c2a2
3) CTRL: the right to modify the access assignments themselves, 3c2a3

to the same classes of users to which file access can be assigned, 3c2b

The access assignments for the directory are stored in the directory's "access descriptor", identical in form to a file's access descriptor, The access descriptor is specified initially when the directory is created, and can be modified any time thereafter by anyone with controlling access to the directory, 3c2c

Directory Descriptors 3c3

Associated with every directory is a data structure called a "directory descriptor", which contains information about the directory, and which has the following format: 3c3a

```
DIRDSC* ==> LIST (                                                3c3a1

   <crtor>  %file creator% CHARSTR,                               3c3a1a
   <accdsc> %access descriptor% ACCDSC*)                          3c3a1b
```

Identifying the Invoking Process 3d

The local process identifies the invoking process, for purposes of enforcing access controls for the directory itself, and for files within it, whenever the directory is opened. The user associated with the invoking process is taken to be, for purposes of establishing or refuting his creatorship of the directory, or of files within it, that specified in the most recent invocation of EXEC's LOGIN procedure. The invoking process is identified as a directory member if it supplies the proper directory password in the OPNDIR procedure. 3d1

Some Similarities 3e

Files and directories bear a striking similarity to data stores and packages, respectively. The similarity is so strong that we define a shorthand for denoting an element of the file: 3e1

```
FSELECTOR* ==> DSELECTOR*                                         3e1a
```

with FILENAME and DID, substituted in the definition for data store key and PKH, respectively. 3e2

We define the following shorthand to denote a filename FILENAME, qualified by the directory DID that contains it: 3e3

```
FILE* ==> LIST (%did% INTEGER, %filename% CHARSTR)                3e3a
```

and we define a list of files as: 3e4

```
FILELIST* ==> LIST (%filename% CHARSTR, ...)                      3e4a
```

PROCEDURES 4

Directory manipulation 4a

Open directory 4a1

OPNDIR (dirname, password => did) 4a1a

Provided the invoking process has read access to the directory, this procedure opens the local process's directory DIRNAME, and makes it known to the invoking process via the handle DID. 4a1b

If PASSWORD is specified (correctly), the user associated with the invoking process is identified as a directory member (a fact considered in subsequent file access control checks). 4a1c

Argument/result types: 4a1d

```
dirname = CHARSTR                 4a1d1
password= CHARSTR / EMPTY         4a1d2
did     = INTEGER                 4a1d3
```

Close directory 4a2

CLSDIR (did) 4a2a

This procedure closes the local process's previously-opened directory, known via DID, and makes it again unknown. 4a2b

Argument/result types: 4a2c

did= INTEGER 4a2c1

List directory 4a3

LSTDIR (did, dst, dstype => count, value) 4a3a

Provided the invoking process has read access to the directory, this procedure first outputs COUNT the number of files in the directory, then a list of the files in the directory identified by DID in the local process, in the form LIST %filename% CHARSTR, ...), to a destination DST whose nature is specified by DSTYPE: 4a3b

PARM: the list is to be returned to the caller as VALUE (i.e. as a result of the procedure). 4a3b1

FILE: the list is to replace the current value of an element DSTELM of a file in one of the local process's previously-opened directories (implicitly named by DSTELM). 4a3b2

NETC: the list is to be transmitted via a network connection, to socket SOCKET at host HOST, in one of the following formats FORMAT: 4a3b3

PCPFRK: that defined by PCP for IPC of data structures between Tenex forks (a 36-bit connection). 4a3b3a

PCPNET: that defined by PCP for IPC of data structures between ARPANET processes (an 8-bit connection). 4a3b3b

CRLF: (for SEQTXT and RECTXT file elements only) the text of the string, or of each string in the list, terminated by CRLF, appended to the connection's 8-bit byte stream. 4a3b3c

CHNL: the list is transmitted via the PCP channel identified by the local process port handle PORH. (Ports and channels are discussed in the Process Management Package document.) 4a3b4

Note that the actual format of the data transmitted between processes is documented in "PCP Data Structure Formats (PCPFMT)". 4a3c

Argument/result types: 4a3d

```
did    = INTEGER                                                 4a3d1
dstype= INTEGER [PARM=0 / FILE=1 / NETC=2 / CHNL=3]              4a3d2
   PARM: dst= EMPTY                                              4a3d2a
   FILE: dst= %dstelm% FSELECTOR*                                4a3d2b
   NETC: dst= LIST (%host% INTEGER, %socket% INTEGER,
              %format% INTEGER [PCPFRK=0 / PCPNET=1 /
              CRLF=2])                                           4a3d2c
   CHNL: dst= %porh% INTEGER                                     4a3d2d
count = INTEGER                                                  4a3d3
value = FILELIST* / EMPTY                                        4a3d4
```

File manipulation 4b

Create file 4b1

CRTFIL (did, count, filelist, accdsc) 4b1a

Provided that the invoking process has write access to the directory DID, this procedure creates a list of COUNT files FILELIST with access descriptor ACCDSC in the directory specified by DID in the local process. Note that the initial content of this file is EMPTY. 4b1b

Argument/result types: 4b1c

did - INTEGER 4b1c1
count - INTEGER 4b1c2
filelist - FILELIST* 4b1c3
accdsc - ACCDSC* 4b1c4

Delete file 4b2

DELFIL (did, count, filelist) 4b2a

Provided the invoking process has write access to the files in FILELIST, and write access to the directory identified by DID, this procedure deletes the COUNT files from the local process. 4b2b

Argument/result types: 4b2c

did - INTEGER 4b2c1
count - INTEGER 4b2c2
filelist - FILELIST* 4b2c3

Rename file 4b3

RENFIL (src-did, count, src-filelist, dst-did, dst-filelist) 4b3a

Provided the invoking process has write access to the files in the source file list SRC-FILELIST and both the directories specified by SRC-DID and DST-DID, this procedure renames the source files to be the destination files DST-FILELIST. 4b3b

Argument/result types: 4b3c

src-did - INTEGER 4b3c1
count - INTEGER 4b3c2

src-filelist = FILELIST* 4b3c3
dst-did = INTEGER 4b3c4
dst-filelist = FILELIST* 4b3c5

Get unique file name 4b4

UNQFIL (did, number -> filelist) 4b4a

NUMBER new and unique filenames in directory DID in the local process are returned to the invoking process. 4b4b

Argument/result types: 4b4c

did = INTEGER 4b4c1
number = INTEGER 4b4c2
filelist = FILELIST* 4b4c3

Convert file 4b5

CNVFIL (file, newfile, usetype, alg) 4b5a

Provided the invoking process has read access to the file FILE and write access to the directory for NEWFILE, and that there is a conversion procedure for converting from the use type and structure type of FILE to the use type USETYPE and structure type desired as indicated by the algorithm ALG, the local process will perform the conversion and create the new file NEWFILE. 4b5b

It is expected that the conversion algorithms for tools with use or structure types that at are at all uncommon will be provided by the tool installer. 4b5c

Argument/result types: 4b5d

file = FILE* 4b5d1
newfile = FILE* 4b5d2
usetype = CHARSTR 4b5d3
alg = CHARSTR 4b5d4

File element manipulation 4c

Get file 4c1

GETFIL (fileelm, disp, dst, dstype => value) 4c1a

Provided the invoking process has read access to the file, this procedure outputs a copy of an element FILEELM (which may be the whole file) of a file in one of the local process's previously-opened directories (implicitly named by FILEELM), to a destination DST whose nature is specified by DSTYPE: 4c1b

PARM: the file element is to be returned to the caller as VALUE (i.e. as a result of the procedure). 4c1b1

FILE: the file element is to replace the current value of an element DSTELM of a file in one of the local process's previously-opened directories (implicitly named by DSTELM). The invoking process must have write access to the destination file. 4c1b2

The file element is either replaced by EMPTY (i.e. moved) or left unchanged (copied), according to DISP. To move the element, the invoking process must have write access to the file. 4c1b3

NETC: the file element is to be transmitted via a network connection, to socket SOCKET at host HOST, using format FORMAT (same as for LSTDIR). 4c1b4

CHNL: the file element is transmitted via a PCP channel attached to the port identified by the port handle PROH of the local process. (Channels and ports are discussed in the Process Management Package document.) 4c1b5

Argument/result types: 4c1c

```
fileelm- FSELECTOR*                                              4c1c1
disp   - INTEGER [DELETE=0 / RETAIN=1]                           4c1c2
dstype - INTEGER [PARM=0 / FILE=1 / NETC=2 / CHNL=3]             4c1c3
   PARM: dst- EMPTY                                             4c1c3a
   FILE: dst- %dstelm% FSELECTOR*                               4c1c3b
   NETC: dst- LIST (%host% INTEGER, %socket% INTEGER,
              %format% INTEGER [PCPFRK=0 / PCPNET=1 /
              CRLF=2])                                          4c1c3c
```

```
         CHNL: dst- %porh% INTEGER                                 4c1c3d
      value  - any / EMPTY                                         4c1c4
```

Put file 4c2

PUTFIL (fileelm, disp, src, srctype) 4c2a

Provided the invoking process has write access to the file, this procedure replaces an element FILEELM (which may be the whole file) of a file in one of the local process's previously-opened directories (implicitly named by FILEELM), from a source SRC whose nature is specified by SRCTYPE: 4c2b

PARM: the source is SRC (i.e. an argument of the procedure). 4c2b1

FILE: the source is the current value of an element SRCELM of a file in one of the local process's previously-opened directories (implicitly named by SRCELM). The invoking process must have read access to the source. 4c2b2

The source element is either replaced by EMPTY (i.e. moved) or left unchanged (copied), according to DISP. To move the source element, the invoking process must have write access to the source file. 4c2b3

NETC: the source will be transmitted via a network connection, from socket SOCKET at host HOST, using format FORMAT (same as for LSTDIR). 4c2b4

CHNL: the file element is transmitted via a PCP channel attached to the port identified by the port handle PORH of the local process. (Channels and ports are discussed in the Process Management Package document.) 4c2b5

Argument/result types: 4c2c

```
      fileelm- FSELECTOR*                                          4c2c1
      disp   - INTEGER [DELETE=0 / RETAIN=1]                       4c2c2
      srctype- INTEGER [PARM=0 / FILE=1 / NETC=2 / CHNL=3]         4c2c3
         PARM: src- any                                            4c2c3a
         FILE: src- %srcelm% FSELECTOR*                            4c2c3b
         NETC: src- LIST (%host% INTEGER, %socket% INTEGER,
                    %format% INTEGER [PCPFRK=0 / PCPNET=1 /
                    CRLF=2])                                       4c2c3c
         CHNL: src- %porh% INTEGER                                 4c2c3d
```

Get file structure type 4c3

GETST (fileelm, dst, dstype => value) 4c3a

Provided the invoking process has read access to the file named by FILEELM the structure of that file element is returned as a "prototype" to destination DST as indicated by DSTYPE. That is, a data structure of the same form as the file element is returned, but the content of the data structure is not meaningful and is reduced to the minimum. The possible DSTYPEs are: 4c3b

PARM: the file element structure is to be returned to the caller as VALUE (i.e. as a result of the procedure). 4c3b1

FILE: the file element structure is to replace the current value of a file element in one of the local process's previously-opened directories (implicitly named by DSTELM). The invoking process must have write access to the destination file. 4c3b2

NETC: the file element structure is to be transmitted via a network connection, to socket SOCKET at host HOST, in format FORMAT (see LSTDIR). 4c3b3

CHNL: the file element structure is transmitted via a PCP channel attached to the port identified by the port handle PROH of the local process. (Channels and ports are discussed in the Process Management Package document.) 4c3b4

Argument/result types: 4c3c

```
fileelm  - FSELECTOR*                                        4c3c1
dstype   - INTEGER [PARM=0 / FILE=1 / NETC=2 / CHNL=3]       4c3c2
   PARM: dst- EMPTY                                          4c3c2a
   FILE: dst- %dstelm% FSELECTOR*                            4c3c2b
   NETC: dst- LIST (%host% INTEGER, %socket% INTEGER,
              %format% INTEGER [PCPFRK=0 / PCPNET=1 /
              CRLF=2])                                       4c3c2c
   CHNL: dst- %porh% INTEGER                                 4c3c2d
value    - LIST (%filename% CHARSTR, ...) / EMPTY            4c3c3
```

Delete file element 4c4

DELELM (fileelm) 4c4a

Provided the invoking process has write access to the file, this procedure replaces an element FILEELM of a file in one of the local process's previously-opened directories (implicitly named by FILEELM) with EMPTY. 4c4b

Argument/result types: 4c4c

fileelm= FSELECTOR* 4c4c1

DATA STORES 5

DESCS List of directory and file descriptors 5a

This data store is a list of the directory descriptors DIRDSCs, and file descriptors FILDSCS for all files FILENAMEs, for all open directories DIDs with names DIRNAMEs within the local process. It also contains for each directory, the user USER who opened it, and his relationship REL to it. The data store is read-only, except for the ACCDSC field of each directory and file descriptor, which can be written by anyone with controlling access to the directory or file. 5a1

Data structure type: 5a2

```
<descs> LIST (<dirname> LIST (%did% INTEGER, %dirdsc%
        DIRDSC*, %usedsc% LIST (%user% CHARSTR, %rel%
        INTEGER [CRT=0 / MEM=1 / PUB=2]), %fildscs% LIST
        (<filename> %fildsc% FILDSC*, ...)), ...)          5a2a
```

The File Package
FP Version 2

22-NOV-74

Jon Postel
Augmentation Research Center

Stanford Research Institute
Menlo Park, California 94025

The File Package (FP) is a file manipulation tool that operates within the setting provided by the Procedure Call Protocol (PCP -- 24459,), with which the reader of the present document is assumed familiar, FILE 2 / The File Package

(J24582) 22-NOV-74 16:59;;;; Title: Author(s): Jonathan B. Postel/JBP; Sub-Collections: SRI-ARC; Clerk: JBP; Origin: < POSTEL, NSW-FILE.NLS;18, >, 22-NOV-74 15:53 JBP ;;;;####;

BATCH 2 / The Batch Job Package

1

PREFACE 2

The Procedure Call Protocol (PCP) is an inter-process and/or inter-host protocol that permits a collection of processes within one or more ARPANET hosts to communicate at the procedure call level. In effect, it makes the component procedures of remote software systems as accessible to the programmer as those within his own system. PCP specifies both a virtual programming environment (VPE) in which remote procedures may be assumed to operate, as well as the inter-process exchanges that implement it. 2a

The Multi-Process Software System (MPSS) whose construction PCP makes practical and of which the NSW is an example, consists of collections of "procedures" and "data stores" called "packages", in one or more "processes", interconnected in a tree structure by "physical channels". Procedures within a process have free access to the procedures (and data stores) of each process adjacent to it in the tree structure, and may call upon them as if they were local subroutines. Superimposed upon the tree structure is a more general set of interconnections which give non-adjacent processes in the tree the same kind of access to one another. 2b

The MPSS is implemented by: 2c

1) low-level protocols which provide the basic, inter-process communicaton (IPC) facilities by which channels are implemented: an inter-host IPC protocol (PCPHST), an inter-Tenex-fork IPC protocol (PCPFRK), and data structure format specifications for both connection types (PCPFMT). 2c1

2) PCP proper, which largely defines the VPE (especially, the procedure call and return mechanism) and specifies the inter-process control exchanges required to implement it. 2c2

3) a set of system packages, implemented within each process, which augment PCP proper by providing mechanisms by which user procedures can: call remote procedures (implemented by the Procedure Interface Package, PIP), manipulate remote data stores (implemented by the PCP Support Package, PSP), and interconnect processes (implemented by the Process Management Package, PMP). 2c3

4) user packages in each process. 2c4

INTRODUCTION 3

The Batch Job Package (package name=BJP) contains those procedures and data stores which a remote process requires to employ the batch processing services of this host, The package contains procedures for creating and deleting batch jobs, for retrieving or altering the status of a batch job, for controlling the transmission of its input/output streams, and for communicating with the batch system's operator, 3a

This package is only implemented at a host that actually provides a batch processing facility, 3b

PROCEDURES 4

Create batch job 4a

CRTJOB (infiles, outfiles => jobid) 4a1

This procedure queues a job for processing by the local process's batch system, and returns the job identifier JOBID by which the job is thereafter known, 4a2

The procedure will retrieve the job's input files INFILES, schedule the job for execution, and eventually return its output files as requested by OUTFILES, 4a3

The batch input/output stream to which each file corresponds is identified by STRMNAME, The following universal stream names are defined (but not necessarily accepted by every local process); other stream names may be defined and accepted by a particular host process: 4a4

CRD: the job's primary card (input) stream, 4a4a
PRT: the job's primary print (output) stream, and 4a4b
PUN: the job's primary punch (output) stream; 4a4c

The local process is to retrieve/save each input/output file by using the parameters supplied in the INFILES/OUTFILES argument to make calls to the appropriate file packages, 4a5

Argument/result types: 4a6

```
infiles = LIST (<strmname> src, ...)                 4a6a
outfiles= LIST (<strmname> dst, ...)                 4a6b
```

src/dst - LIST (%host% INTEGER, %account-designator% LIST (%user% CHARSTR, %password% CHARSTR, %acct% CHARSTR), %workspace-designator% LIST (%dirname% CHARSTR, %password% CHARSTR), %fileelm% FSELECTOR*, %disp% INTEGER, 4a6c

jobid - INTEGER 4a6d

Delete batch job 4b

DELJOB (jobid) 4b1

This procedure deletes the previously-created batch job identified by JOBID. Any input/output files that have yet to be retrieved/returned are ignored/discarded. 4b2

Argument/result types: 4b3

jobid= INTEGER 4b3a

Cancel batch job 4c

CANJOB (jobid) 4c1

This procedure cancels the execution phase (interrupting the job's execution if necessary) of the previously-created job identified by JOBID. The job remains in the batch system's queue, and any output files generated by the job before its cancellation will be disposed of as previously specified. 4c2

Argument/result types: 4c3

jobid= INTEGER 4c3a

Retrieve batch job status 4d

STSJOB (jobid => status) 4d1

This procedure returns the status STATUS of the job identified by JOBID. The exact format and semantics of the status information are yet to be determined. 4d2

Argument/result types: 4d3

jobid = INTEGER 4d3a

status= LIST (CHARSTR, ...) 4d3b

Modify batch job 4e

MODJOB (jobid, parms) 4e1

This procedure modifies, in a host-dependent way, the parameters PARMS of the of previously-created job identified by JOBID. 4e2

Argument/result types: 4e3

```
jobid= INTEGER                                                        4e3a
parms= any                                                            4e3b
```

Query batch system operator 4f

QRYOPR (message, rsvp => reply) 4f1

This procedure transmits message MESSAGE to the batch system's operator, and, if RSVP is TRUE, returns his reply REPLY. 4f2

Argument/result types: 4f3

```
message= CHARSTR                                                      4f3a
rsvp   = BOOLEAN                                                      4f3b
reply  = CHARSTR / EMPTY                                              4f3c
```

Execute remote-operator command 4g

EXECMD (command => response) 4g1

This procedure executes the host-dependent remote-operator command COMMAND, and returns the batch system's response to it. 4g2

Argument/result types: 4g3

```
command = CHARSTR                                                     4g3a
response= LIST (CHARSTR, ...)                                         4g3b
```

DATA STORES 5

This package contains no data stores. 5a

The Batch Job Package
BJP Version 2

22-NOV-74

Jon Postel
Augmentation Research Center

Stanford Research Institute
Menlo Park, California 94025

The Batch Job Package operates within the setting provided by the Procedure Call Protocol (PCP -- 24459,), with which the reader of the present document is assumed familiar.

BATCH 2 / The Batch Job Package

(J24583) 22-NOV-74 17:01;;;; Title: Author(s): Jonathan B. Postel/JBP; Sub-Collections: SRI-ARC; Clerk: JBP; Origin: < POSTEL, NSW-BATCH.NLS;6, >, 22-NOV-74 16:22 JBP ;;;;####;

BOXES 2 / Black Boxes in PCP

; 1

JBP 22 NOV 74 7:53PM

JBP 22-NOV-74 17:03 24584
Black Boxes in PCP

Introduction 2

The various black boxes described in "NSW Black Boxes" by Millstein and Warshall of 1-Oct-74 [CADD-7410-0112] are cast in PCP calls. The intent here is to show how the functions described in the black boxes document could be implemented using procedures as defined in the PCP series of documents. This is not intended as a specification of the actual implementation of these functions but only to further the understanding of procedure call protocol. 2a

Data structures 3

Introduction 3a

These data structure definitions parallel the definitions in the black boxes document, and are used in the remainder of this document to act as a shorthand in passing procedure arguments and results. 3a1

Account designator 3b

%acd% LIST (%user%CHARSTR, %password%CHARSTR, %account%CHARSTR) 3b1

Host 3c

%host%INTEGER 3c1

Workspace designator 3d

%wsd% LIST (%directory%CHARSTR, %password%CHARSTR) 3d1

Filename 3e

%filename% CHARSTR 3e1

Filelist 3f

%filelist% LIST (%filename%CHARSTR, ...) 3f1

Filepairs 3g

%filepairs% LIST (LIST (%src-filename%CHARSTR, %dst-filename%CHARSTR), ...) 3g1

File-id 3h

%file-id% LIST (%host%INTEGER, wsd, %filename%CHARSTR) 3h1

Cost 3i

%cost%INTEGER 3i1

Condition/error code 3j

Every pcp return includes a condition or error code (see PCP or PIP). 3j1

Procedures 4

Introduction 4a

Shown in this section is an example implementation of the black box functions. We would expect that the Works Manager would initially create processes and open the appropriate packages in each tool bearing host. This initialization is shown in the Prologue, and for completeness the closing of these packages and processes is shown in an Epilogue. 4a1

The notation here is a slight simplification of the required message format to: 4a2

```
Call ( ph, pk, procname (arguments & results))        4a2a

    where:                                            4a2a1

        ph       = process handle                     4a2a1a
        pk       = package handle                     4a2a1b
        procname = procedure name                     4a2a1c
```

Prologue 4b

```
INITIALIZE (host, "file-package" -> ph, epk, fpk)     4b1

    Once                                              4b1a

        Call (self, 0,    OPNPKS (PMP => mpk))        4b1a1

    For each host                                     4b1b

        Call (self, mpk,  CRTPRC ($N host socket => Ph))   4b1b1
        Call (ph,   0,    OPNPKS (FP, EXEC -> fpk, ePk))   4b1b2
        Call (ph,   epk,  LOGIN (acd))                     4b1b3
```

```
Epilogue                                                              4c

   COMPLETE (ph)                                                      4c1

      For each host                                                   4c1a

         Call (self, mpk,   DELPRC (ph -> cost))                      4c1a1

Net file copy                                                         4d

   NETCOPY (sph, sfpk, src-wsd, src-filename, dph, dfpk, dst-wsd,
   access -> dst-filename)                                            4d1

      Call (sph,  sfpk,  OPNDIR (src-wsd -> src-did))                 4d1a
      Call (dph,  dfpk,  OPNDIR (dst-wsd -> dst-did))                 4d1b
      Call (dph,  dfpk,  UNQFIL (dst-did, "1" -> dst-filename))       4d1c
      Call (dph,  dfpk,  CRTFIL (dst-did, "1", dst-filename,
      access))                                                        4d1d
      Call (self, mpk,   CRTPHYCHN (sph, dph -> sporh, dporh,
      pch))                                                           4d1e
      Call (dph,  dfpk,  PUTFIL ((dst-did, dst-filename),
      "retain", dporh, "chnl"))                                       4d1f
      Call (sph,  sfpk,  GETFIL ((src-did, src-filename),
      "retain", sporh, "chnl"))                                       4d1g
      Call (self, mpk,   DELPHYCHN (pch)                              4d1h
      Call (sph,  sfpk,  CLSDIR (src-did))                            4d1i
      Call (dph,  dfpk,  CLSDIR (dst-did))                            4d1j

Local file copy                                                       4e

   LOCALCOPY (ph, fpk, src-wsd, src-filename, dst-wsd, access ->
   dst-filename)                                                      4e1

      Call (ph,   fpk,   OPNDIR (src-wsd -> src-did))                 4e1a
      Call (ph,   fpk,   OPNDIR (dst-wsd -> dst-did))                 4e1b
      Call (ph,   fpk,   UNQFIL (dst-did, "1" -> dst-filename))       4e1c
      Call (ph,   fpk,   CRTFIL (dst-did, "1", dst-filename,
      access))                                                        4e1d
      Call (ph,   fpk,   PUTFIL ((dst-did, dst-filename),
      "retain", (src-did, src-filename), "file"))                     4e1e
      Call (ph,   fpk,   CLSDIR (src-did))                            4e1f
      Call (ph,   fpk,   CLSDIR (dst-did))                            4e1g
```

```
Delete file                                                        4f

   DELETEFILE (ph, fpk, wsd, filename)                             4f1

      Call (ph,   fpk,   OPNDIR (wsd => did))                      4f1a
      Call (ph,   fpk,   DELFIL (did, "1", filename))              4f1b
      Call (ph,   fpk,   CLSDIR (did))                             4f1c

Delete all files                                                   4g

   DELETEWS (ph, fpk, wsd)                                         4g1

      Call (ph,   fpk,   OPNDIR (wsd => did))                      4g1a
      Call (ph,   fpk,   LSTDIR (did, EMPTY, "parm" => count,
      filelist))                                                   4g1b
      Call (ph,   fpk,   DELFIL (did, count, filelist))            4g1c
      Call (dph,  dfpk,  CLSDIR (did))                             4g1d

Local file move                                                    4h

   LOCALMOVE (ph, fpk, src-wsd, src-filename, dst-wsd =>
   dst-filename)                                                   4h1

      Call (ph,   fpk,   OPNDIR (src-wsd => src-did))              4h1a
      Call (ph,   fpk,   OPNDIR (dst-wsd => dst-did))              4h1b
      Call (ph,   fpk,   UNQFIL (dst-did, "1" => dst-filename))    4h1c
      Call (ph,   fpk,   RENFIL (src-did, "1", src-filename,
      dst-did, dst-filename))                                      4h1d
      Call (ph,   fpk,   CLSDIR (src-did))                         4h1e
      Call (ph,   fpk,   CLSDIR (dst-did))                         4h1f

Move workspace                                                     4i

   MOVEWS (ph, fpk, src-wsd,  dst-wsd, access => filepairs)        4i1
      Call (ph,   fpk,   OPNDIR (src-wsd => did))                  4i1a
      Call (ph,   fpk,   LSTDIR (did, EMPTY, "parm" => count,
      src-filelist))                                               4i1b
      Call (ph,   fpk,   UNQFIL (did , count => dst-filelist))     4i1c
      Call (ph,   fpk,   CRTFIL (did, count, dst-filelist,
      access))                                                     4i1d
      Call (ph,   fpk,   RENFIL (src-did, count, src-filelist,
      dst-did, dst-filelist))                                      4i1e
      Call (ph,   fpk,   CLSDIR (did))                             4i1f
   Get local catalogue                                                    4j

      GETCAT (ph, fpk, wsd => filelist)                                  4j1
         Call (ph,   fpk,   OPNDIR (wsd => did))                        4j1a
         Call (ph,   fpk,   LSTDIR (did, EMPTY, "parm" => count,
         filelist))                                                     4j1b
         Call (ph,   fpk,   CLSDIR (did))                               4j1c

   State probe                                                            4k

      STATE (ph => usage)                                                4k1

         Call (ph,   0,     RDDATA ((self, epk, USAGE) => usage))       4k1a

   Accounting probe                                                       4l

      ACCOUNT (ph => cents)                                              4l1

         Call (ph,   0,     RDDATA ((self, epk, COST) => cents))        4l1a
```

Appendix 5

Introduction 5a

In this appendix is presented a possible implementation of the black box functions using the procedures defined in the PCP documents. This is not the recommended implementation but is shown only to promote an understanding of the procedure call protocol. 5a1

Net file copy 5b

NETCOPY (src-acd, src-host, src-wsd, src-filename, dst-acd, dst-host, dst-wsd, access -> dst-filename) 5b1

```
Call (self, 0,     OPNPKS (PMP -> mpk))                                5b1a
Call (self, mpk,   CRTPRC ($N src-host socket -> sph))                 5b1b
Call (self, mpk,   CRTPRC ($N dst-host socket -> dph))                 5b1c
Call (sph,  0,     OPNPKS (FP, EXEC -> sfpk, sepk))                    5b1d
Call (dph,  0,     OPNPKS (FP, EXEC -> dfpk, depk))                    5b1e
Call (sph,  sepk,  LOGIN (src-acd))                                    5b1f
Call (dph,  depk,  LOGIN (dst-acd))                                    5b1g
Call (sph,  sfpk,  OPNDIR (src-wsd -> src-did))                        5b1h
Call (dph,  dfpk,  OPNDIR (dst-wsd -> dst-did))                        5b1i
Call (dph,  dfpk,  UNQFIL (dst-did, "1" -> dst-filename))              5b1j
Call (dph,  dfpk,  CRTFIL (dst-did, "1", dst-filename,
access))                                                               5b1k
Call (self, mpk,   CRTPHYCHN (sph, dph -> sporh, dporh,
phc))                                                                  5b1l
Call (dph,  dfpk,  PUTFIL ((dst-did, dst-filename),
"retain", dporh, "chnl"))                                              5b1m
Call (sph,  sfpk,  GETFIL ((src-did, src-filename),
"retain", sporh, "chnl"))                                              5b1n
Call (self, mpk,   DELPHYCHN (phc)                                     5b1o
Call (self, mpk,   DELPRC (sph -> scost))                              5b1p
Call (self, mpk,   DELPRC (dph -> dcost))                              5b1q
```

Local file copy 5c

LOCALCOPY (acd, host, src-wsd, src-filename, dst-wsd, access -> dst-filename) 5c1

```
Call (self, 0,     OPNPKS (PMP -> mpk))                                5c1a
Call (self, mpk,   CRTPRC ($N host socket -> ph))                      5c1b
Call (ph,   0,     OPNPKS (FP, EXEC -> fpk, epk))                      5c1c
Call (ph,   epk,   LOGIN (acd))                                        5c1d
Call (ph,   fpk,   OPNDIR (src-wsd -> src-did))                        5c1e
Call (ph,   fpk,   OPNDIR (dst-wsd -> dst-did))                        5c1f
Call (ph,   fpk,   UNQFIL (dst-did, "1" -> dst-filename))              5c1g
```

```
   Call (ph,   fpk,   CRTFIL (dst-did, "1", dst-filename,
   access))                                                   5c1h
   Call (ph,   fpk,   PUTFIL ((dst-did, dst-filename),
   "retain", (src-did, src-filename), "file"))                5c1i
   Call (self, mpk,   DELPRC (ph => cost))                        5c1j
```

Delete file 5d

DELETEFILE (acd, host, wsd, filename => filename) 5d1

```
   Call (self, 0,     OPNPKS (PMP => mpk))                        5d1a
   Call (self, mpk,   CRTPRC ($N host socket => ph))              5d1b
   Call (ph,   0,     OPNPKS (FP, EXEC => fpk, epk))              5d1c
   Call (ph,   epk,   LOGIN (acd))                                5d1d
   Call (ph,   fpk,   OPNDIR (wsd => did))                        5d1e
   Call (ph,   fpk,   DELFIL (did, "1", filename))                5d1f
   Call (self, mpk,   DELPRC (ph => cost))                        5d1g
```

Delete all files 5e

DELETEWS (acd, host, wsd) 5e1

```
   Call (self, 0,     OPNPKS (PMP => mpk)),                       5e1a
   Call (self, mpk,   CRTPRC ($N host socket => ph))              5e1b
   Call (ph,   0,     OPNPKS (FP, EXEC => fpk, epk))              5e1c
   Call (ph,   epk,   LOGIN (acd))                                5e1d
   Call (ph,   fpk,   OPNDIR (wsd => did))                        5e1e
   Call (ph,   fpk,   LSTDIR (did, EMPTY, "parm" => count,
   filelist))                                                 5e1f
   Call (ph,   fpk,   DELFIL (did, count, filelist))              5e1g
   Call (self, mpk,   DELPRC (ph => cost))                        5e1h
```

Local file move 5f

LOCALMOVE (acd, host, src-wsd, src-filename, dst-wsd => dst-filename) 5f1

```
   Call (self, 0,     OPNPKS (PMP => mpk))                        5f1a
   Call (self, mpk,   CRTPRC ($N host socket => ph))              5f1b
   Call (ph,   0,     OPNPKS (FP, EXEC => fpk, epk))              5f1c
   Call (ph,   epk,   LOGIN (acd))                                5f1d
   Call (ph,   fpk,   OPNDIR (src-wsd => src-did))                5f1e
   Call (ph,   fpk,   OPNDIR (dst-wsd => dst-did))                5f1f
   Call (ph,   fpk,   UNQFIL (dst-did, "1" => dst-filename))      5f1g
   Call (ph,   fpk,   RENFIL (src-did, "1", src-filename,
   dst-did, dst-filename))                                    5f1h
   Call (self, mpk,   DELPRC (ph => cost))                        5f1i
```

Move workspace 5g

```
MOVEWS (acd, host, src-wsd,  dst-wsd, access -> filepairs))  5g1

   Call (self, 0,      OPNPKS (PMP -> mpk))                  5g1a
   Call (self, mpk,    CRTPRC ($N host socket -> ph))        5g1b
   Call (ph,   0,      OPNPKS (FP, EXEC -> fpk, epk))        5g1c
   Call (ph,   epk,    LOGIN (acd))                          5g1d
   Call (ph,   fpk,    OPNDIR (src-wsd -> did))              5g1e
   Call (ph,   fpk,    LSTDIR (did, EMPTY, "parm" -> count,
   src-filelist))                                            5g1f
   Call (ph,   fpk,    UNQFIL (did , count -> dst-filelist)) 5g1g
   Call (ph,   fpk,    CRTFIL (did, count, dst-filelist,
   access))                                                  5g1h
   Call (ph,   fpk,    RENFIL (src-did, count, src-filelist,
   dst-did, dst-filelist))                                   5g1i
   Call (self, mpk,    DELPRC (ph -> cost))                  5g1j
```

Get local catalogue 5h

```
GETCAT (acd, host, wsd -> filelist))                         5h1

   Call (self, 0,      OPNPKS (PMP -> mpk))                  5h1a
   Call (self, mpk,    CRTPRC ($N host socket -> ph))        5h1b
   Call (ph,   0,      OPNPKS (FP, EXEC -> fpk, epk))        5h1c
   Call (ph,   epk,    LOGIN (acd))                          5h1d
   Call (ph,   fpk,    OPNDIR (wsd -> did))                  5h1e
   Call (ph,   fpk,    LSTDIR (did, EMPTY, "parm" -> count,
   filelist))                                                5h1f
   Call (self, mpk,    DELPRC (ph -> cost))                  5h1g
```

State probe 5i

```
STATE (acd, host -> usage)                                   5i1

   Call (self, 0,      OPNPKS (PMP -> mpk))                  5i1a
   Call (self, mpk,    CRTPRC ($N host socket -> ph))        5i1b
   Call (ph,   0,      OPNPKS (EXEC -> epk))                 5i1c
   Call (ph,   epk,    LOGIN (acd))                          5i1d
   ...                                                       5i1e
   Call (ph,   0,      RDDATA ((self, epk, USAGE) -> usage)) 5i1f
   ...                                                       5i1g
   Call (self, mpk,    DELPRC (ph -> cost))                  5i1h
```

Accounting probe 5j

ACCOUNT (acd, host => cents) 5j1

```
Call (self, 0,     OPNPKS (PMP => mpk))                        5j1a
Call (self, mpk,   CRTPRC (SN host socket => ph))              5j1b
Call (ph,   0,     OPNPKS (EXEC => epk))                       5j1c
Call (ph,   epk,   LOGIN (acd))                                5j1d
...                                                            5j1e
Call (ph,   0,     RDDATA ((self, epk, COST) => cents))        5j1f
...                                                            5j1g
Call (self, mpk,   DELPRC (ph => cost))                        5j1h
```

Black Boxes in PCP
Version 2

22-NOV-74

Jon Postel
Augmentation Research Center

Stanford Research Institute
Menlo Park, California 94025

This document describes the mapping of the Black Boxes described by Millstein & Warshall into the procedure calls defined by White & Postel,

(J24584) 22-NOV-74 17:03;;;; Title: Author(s): Jonathan B. Postel/JBP; Sub-Collections: SRI-ARC; Clerk: JBP; Origin: < POSTEL, NSW-BLACK-BOXES.NLS;13, >, 22-NOV-74 16:28 JBP ;;;;####;

DCE 23-NOV-74 10:56 24585

To Placko re wider distribution of his (31374,) paper

Mike: Just to say that I read and appreciatd your item, "Notes On The Application Of The Arc Utility At SRI", (GJOURNAL, 31374,). Nicely written, good coverage -- PLUS, building up important dialogue base in recorded form. I'd like to seem more of our interested parties have an [info only] citation -- esp. jcn, jhb, and rll in our Utility group; also it would seem that all of our architects would be interested, and I personally don't see anything in the content that would deter me from sharing it with any of them. Up to you, of course. 1

To Placko re wider distribution of his (31374,) paper

(J24585) 23-NOV-74 10:56;;;; Title: Author(s): Douglas C. Engelbart/DCE; Distribution: /MAP2([INFO-ONLY]) ; Sub-Collections: SRI-ARC; Clerk: DCE;

To Belleville re his report on ASME CAD session (24573,)

Bob: Just to say that I read and appreciatd your item, "Report On a Presentation to the ASME (Amer. Soc. of Mechanical Eng)", (GJOURNAL, 24573,). It was nicely written and had good coverage -- PLUS, building up important dialogue base in recorded form. I'm putting a copy of the memo in our "marketing" file, relating to future evolution of AKW working relationship with the CAD world; and I'm also looking forward to some good discussions with you on that topic soon.. 1

To Belleville re his report on ASME CAD session (24573,)

(J24586) 23-NOV-74 11:07;;;; Title: Author(s): Douglas C. Engelbart/DCE; Distribution: /RLB2([INFO-ONLY]) ; Sub-Collections: SRI-ARC; Clerk: DCE;

DCE 23-NOV-74 11:07 24587

To Belleville re his report on ASME CAD session (31374,)

Bob: Just to say that I read and appreciatd your item, "Report On a Presentation to the ASME (Amer. Soc. of Mechanical Eng)", (GJOURNAL, 24573,). It was nicely written and had good coverage -- PLUS, building up important dialogue base in recorded form. I'm putting a copy of the memo in our "marketing" file, relating to future evolution of AKW working relationship with the CAD world; and I'm also looking forward to some good discussions with you on that topic soon.. 1

To Belleville re his report on ASME CAD session (31374,)

(J24587) 23-NOV-74 11:07;;;; Title: Author(s): Douglas C. Engelbart/DCE; Distribution: /RLB2([INFO-ONLY]) ; Sub-Collections: SRI-ARC; Clerk: DCE;

SNDMSG Copy: To Russell, re, ANET experiences for ARC/NLS

This responded to Russell's query, on Lukasik's behalf, for comments about our Net experience (grist for Lukasik's Dec talk, same meeting where Dick will talk apparently).

DCE 23-NOV-74 11:25 24588

SNDMSG Copy: To Russell, re. ANET experiences for ARC/NLS

Dave: I take it that you want grist from me, bearing upon usage experience with the ARPANET, especially from ARC's experience. I'm recounting some highlights in narrative style; let me know if you want more items, or more details. 1

In the very earliest days of developing a Network Information Center (NIC), I found an almost universal image in each PI that the documentation on his systems was in embarrasingly poor shape; quite evidently a threat to him, in exposing this inadequacy by opening his resources to remote users. 2

In providing extensive services since the very earliest Net days, our NIC learned how much harder it is to serve the users of a large Net than it is to serve local users of one's own center. 2a

Important point abot Nets, then, is that a new level of quality is required in formal user services (documentation, training, bug-reporting, advice, etc.). 2b

We also learned that a new level of quality is required in the technical-system service; very noticeable lower tolerance to delays, outage, bugs, etc. Some due to greater inconvenience to remote users if they are cut off and can't easily find out what is happening. Some is due to the lack of personal contact -- user and server, not knowing each other personally, don't have empathy for each other's problems. 3

Earliest remarkable observation about ARPANET Community has to do with impact on cooperativeness, working style, etc. Common problems among developers brought people together, and the Net's communication facilities even in the early days (shared files, TTY linking) made collaboration easier. 4

Among the various research groups, particularly for the emerging fellows who cut their teeth on Net projects, there was a marked change over a period of a few years in their acceptance of other's styles and ideas, and in their willingness to cooperate. 4a

To bolster this, earliest services developed in the NIC were to support the collaborative flow of communications: memos, messages, etc.; human Information Agents and Liaison assignments; and the IDENT system that both helped distribute the communications and aided people to locate people. 4b

Our continuing experience in providing heavily knowledge-oriented service over the Net constantly reaffirms how important it is to give special liaison, service, and/or training assignements to local humans who have real identification with the served users.

It is also important to support their work via special communicationss facilities. 4c

The first time we experienced the real power of a Network was in 1970 -- bootstrapping NLS and its support systems from our SDS-940 to the present PDP-10. Our software is all in structured form, generated, stored, manipulated (and now debugged) within the NLS "software workshop." We used our full kit of tools in the 940 to prepare the new source code; shipped files across to Utah's PDP-10 to debug. Programmer flipped back and forth between NLS source-code work, compiler, debugger -- back and forth betweeen machines -- from his same display console, with very nearly the same ease as when working on one machine. We set some sort of record for minimized conversion (and upgrading) effort. 5

The next peak experience was in 1972, when we got DNLS working over the Net using IMLAC terminals (self-contained mini computer programmed to handle the 2-dimensional display interaction, using DNLS core processes in our host). 6

Before that, remote NLS users were all on typewriters, not a particularly demanding use of ANET capability, and not really that convincing for requiring a net as oppossed to dialup phone service. But here we got really quite acceptable level of interactivity, with DNLS's apecial two-dimensional text displays -- a service that would be very expensive to provide if by wide-band private data lines. 6a

First instances then of what we call "shared-screen dialogue", between people at ARC in Menlo Park and at RADC in Rome, New York, working on highly interactive screens where each could point to and control, simultaneously talking on the telephone -- like sharing a blackboard. As far as I am concerned, that is one of the key portents of what the Net can provide. 6b

The $20K price tag on an IMLAC is discouraging. Our $2500 Line Processor device turns any suitable, late-model, high-speed typewriter-like CRT display into a two-dimensional DNLS terminal. Apparently these, when using 4800-baud modems on private wires into the TIPS, are the first cases where the TIPS are being connected to in this fashion. Some technical problems that weren't uncovered before. 7

ALso, we are finding that the Net, via a TIP port, really doesn't deliver burst bandwidth as advertised, at least through very many intermediate-IMP hops. The problem doesn't seem to hit the file-transfer use, so we think it probably bears upon buffer sizes in TIPs. 7a

We feel that the Network's steady influence upon resource sharing, upon multiple-host "tool systems," etc., is having a significant impact upon the concepts and practices of system design. Inter-process protocols, Control Meta Languages, Frontend-Backend splits, etc. seem basic and important. The NSW Program is very important in this respect. We expect that the Intelligent Terminal Program should build upon this approach. 8

RADC undertook a technology-transfer experiment using NLS; three years ago they began experimenting with typewriters through the NET. They bought five IMLACs when they got their own TIP. They now use five slots on OFFICE-1 relatively heavily. Among the recent extensions in application area has been toward heavy-document publication. Have developed considerable project-management usage; branching into support to software engineers (and have begin to contribute to NSW Program). 9

Technology transfer, at least in information-processing technology, is uniquely aided by the Network. For the size and complexity of the new generations of applications systems, user organizations couldn't afford to import them to install in their local computer facilities just for experimentation. The Network very much facilitates the exploratory access, and comparative evaluation. 10

For us, in trying to facilitate a concurrent evolution of knowedge-work augmentation know-how, along with its transfer into the application world, the Network is an absolute necessity. 10a

In the first place, exotic interactive services couldn't be piped into a client's offices practically in any other way. 10b

In the second place, we couldn't run a solid service for such a complex of tools without a contractor like TYMSHARE to support the operating system; and we expect to have NLS service systems running in many different facilities within a few years -- couldn't sensibly plan for this (by a core of people based in a non-profit outfit) without the Network enabling us to maintain the applications software, the documentation, the day-to-day user communications support, etc., from our central workshop terminals. 10c

In the third place, the very tools for supporting collaborative dialogue that are such a basic part of our "augmented knowledge workshop" services, serve a key role in this whole transfer process. Close dialogue between developers, documenters, trainers, user-representative architects and manager-buyers, users, and systems analysts, is necessary for the coherent evolution of large, complex systems, and also for the sensibly-staged transfer into application organizaions. 10d

The ARPANET Newsletter experience was quite noteworthy. Many people contributed; distributed committee did the editorial work (via net collaboration); computer published for hard-copy distriution; on-line access of "preprints" and final editions. The editorship of the SIG-AI Newsletter, for several years, happened to be in SRI's AI group; they developed and published a number of issues successfully using NLS in this way. 11

The DoD Internetting Study Group made heavy use of NLS from late Aug into Oct 74 to develop final report. Three different committees working on one large report (total perhaps 700 pages): heavy revision, many cycles. Used terminals at SRI-Wash already provided under ARPA spport to SRI Defense ENergy Prject; extra terminals borrowed; SRI loaned offie space; DCA clericals trained and supervised on the job by ARC specialist; RADC skilled clerical supervisor helped first week. 12

(Dave, you can better fill in about nature and dynamics of the Study Group and any benefits from NLS support to the development of the report's contents). 12a

The clerical team, directories, and working methods were set up quickly and easily (fair amount of set-up negotiations and arrangements done via Net dialogue). 12b

SNDMSG Copy: To Russell, re. ANET experiences for ARC/NLS

(J24588) 23-NOV-74 11:25;;;; Title: Author(s): Douglas C. Engelbart/DCE; Distribution: /RWW([INFO-ONLY]) JCN([INFO-ONLY]) ; Sub-Collections: SRI-ARC; Clerk: DCE;

DCE 23-NOV-74 11:43 24589

Question for Dirk re his (GJOURNAL, 24543,)

Dirk: 1

The citation I got: 2

K19-0908 DVN: ASAS
Sent: 19-NOV-74 08:26 (GJOURNAL, 24543, 1)
Note: [INFO-ONLY] 2a

Comments: This is a correction to 24454. 2a1

Three questions about it: 3

1) What does the "ASAS" in the title mean? I checked and it isn't an IDENT. I'd like titles to be more informative. 3a

2) The journal file itself looks very interesting and esoteric, but also something of a private nature. I couldn't find any reader-guide to what, why, etc. of content. Is it perchance a periodic Journalization of a private file of yours where you happened to accidently have me on the distribution list? 3b

3) Your Comment citation to 24454 is very confusiong. (J24454) happens to be Sandy's "...A Spade is a Spade.." message, and your citing it adds to the confusion of this whole Journal entry. 3c

Puzzled recipient -- Doug 4

Question for Dirk re his (GJOURNAL, 24543,)

(J24589) 23-NOV-74 11:43;;;; Title: Author(s): Douglas C. Engelbart/DCE; Distribution: /DVN([INFO-ONLY]) ; Sub-Collections: SRI-ARC; Clerk: DCE;

Version 2 of the Procedure Call Protocol (PCP)

This note announces release of the second published version of the Procedure Call Protocol -- PCP Version 2. Version 2 is SUBSTANTIALLY different than Version 1; it and all intermediate, informally distributed PCP documents are obsoleted by this release. 1

Version 2 consists of the following documents. Each is available on-line in two forms: as an NLS file and as a formatted text file. The Journal number (e.g. 24459) refers to the former, of course, and the pathname (e.g. [SRI-ARC]<NLS>PCP.TXT) to the latter, accessible via FTP using USER=ANONYMOUS and PASSWORD=GUEST (no account required). Hardcopy is being forwarded by US Mail to all those who have expressed an interest in PCP. If you don't receive a copy and would like one of this and/or future releases, send a note to that effect to WHITE@SRI-ARC: 2

PCP (24459,) "The Procedure Call Protocol" 2a

This document describes the virtual programming environment provided by PCP, and the inter-process exchanges that implement it. 2a1

Pathname: [SRI-ARC]<NLS>PCP.TXT 2a2

PIP (24460,) "The Procedure Interface Package" 2b

This document describes a package that runs in the setting provided by PCP and that serves as a procedure-call-level interface to PCP proper. It includes procedures for calling, resuming, interrupting, and aborting remote procedures. 2b1

Pathname: [SRI-ARC]<NLS>PIP.TXT 2b2

PSP (24461,) "The PCP Support Package" 2c

This document describes a package that runs in the setting provided by PCP and that augments PCP proper, largely in the area of data store manipulation. It includes procedures for obtaining access to groups of remote procedures and data stores, manipulating remote data stores, and creating temporary ones. 2c1

Pathname: [SRI-ARC]<NLS>PSP.TXT 2c2

PMP (24462,) "The Process Management Package" 2d

This document describes a package that runs in the setting provided by PCP and that provides the necessary tools for interconnecting two or more processes to form a multi-process system (e.g. NSW). It includes procedures for creating, deleting, logically and physically interconnecting processes, and for allocating and releasing processors. 2d1

Pathname: [SRI-ARC]<NLS>PMP.TXT 2d2

PCPFMT (24576,) "PCP Data Structure Formats" 2e

This document defines formats for PCP data structures, each of which is appropriate for one or more physical channel types. 2e1

Pathname: [SRI-ARC]<NLS>PCPFMT.TXT 2e2

PCPHST (24577,) "PCP ARPANET Inter-Host IPC Implementation" 2f

This document defines an implementation, appropriate for mediating communication between Tenex forks, of the IPC primitives required by PCP. 2f1

Pathname: [SRI-ARC]<NLS>PCPHST.TXT 2f2

PCPFRK (24578,) "PCP Tenex Inter-Fork IPC Implementation" 2g

This document defines an implementation, appropriate for mediating communication between processes on different hosts within the ARPANET, of the IPC primitives required by PCP. 2g1

Pathname: [SRI-ARC]<NLS>PCPFRK.TXT 2g2

The first document, PCP, is the place the interested reader should start. It gives the required motivation for the Protocol and states the substance of the Protocol proper. The reader may then, if he chooses, read the next three documents: PIP, PSP, and PMP. The latter has the most to offer the casual reader; the programmer faced with coding in the PCP environment should read all three. The final few documents -- PCPFMT, PCPHST, and PCPFRK -- are of interest only to the PCP implementer. 3

Version 2 of the Procedure Call Protocol (PCP)

(J24590) 23-NOV-74 16:25;;;; Title: Author(s): James E. (Jim) White/JEW; Distribution: /SRI-ARC([INFO-ONLY]) NSW([INFO-ONLY]) ; Sub-Collections: SRI-ARC NSW; Clerk: JEW; Origin: < WHITE, PCP-COVER.NLS;5, >, 23-NOV-74 16:12 JEW ;;;;####;

Version 2 of NSW protocols

This note announces the release of the second published version of several National Software Works (NSW) protocol documents. This set of documents is labeled Version 2. Version 1, as well as all intermediate, informally distributed NSW documents are obsoleted by this release. 1

Several of these documents specify protocols or procedure packages based on the Procedure Call Protocol (PCP -- 24459,), with which the reader is assumed familiar. 2

These documents are available online in two forms: as journal items indicated by the link number [for example the HOST document is journal item 24581]; and as ASCII text files by the indicated pathname [for example the HOST document is text file HOST.TXT in directory NLS at host SRI-ARC]. The files may be reterived from SRI-ARC using the file transfer user name ANONYMOUS and the password GUEST, no account number is needed. 3

Hardcopy is being forwarded by US Mail to all those who have expressed an interest in NSW protocols.. If you don't receive a copy and would like one of this and/or future releases, send a note to that effect to WHITE@SRI-ARC: 4

The specifications are contained in the following documents: 5

HOST "NSW Host Protocol" (24581,) 5a

This document describes the host level protocol used in the NSW. The protocol is a slightly constrained version of the standard ARPANET host to host protocol. The constraints affect the allocation, RFNM wait, and retransmission policies. 5a1

[SRI-ARC]<NLS>HOST.TXT 5a1a

EXEC "The Executive package" (24580,) 5b

This document describes a package that runs in the setting provided by PCP. It includes procedures and data stores for user identification, accounting, and usage information. 5b1

[SRI-ARC]<NLS>EXEC.TXT 5b1a

FILE "The File Package" (24582,) 5c

This document describes a package that runs in the setting provided by PCP. It includes procedures and data stores for opening, closing, and listing directories, for creating, deleting, and renaming files, and for transfering files and file elements between processes. 5c1

[SRI-ARC]<NLS>FILE.TXT 5c1a

BATCH "The Batch Job Package" (24583,) 5d

This document describes a package that runs in the setting provided by PCP. It includes procedures for creating and deleting batch jobs, obtaining the status of a batch job, and communicating with the operator of a batch processing host. This package is implemented at the host that provides the batch processing facility. 5d1

[SRI-ARC]<NLS>BATCH.TXT 5d1a

LLDBUG "The Low-Level Debug Package" (24579,) 5e

This document describes a package that runs in the setting provided by PCP. It includes procedures for a remote process to debug at the assembly-language level, any process known to the local process. The package contains procedures for manipulating and searching the process' address space, for manipulating and searching its symbol tables, and for setting and removing breakpoints from its address space. Its data stores hold process characteristics and state information, and the contents of program symbol tables. 5e1

[SRI-ARC]<NLS>LLDBUG.TXT 5e1a

BOXES "Black Boxes in PCP" (24584,) 5f

This document describes the transliteration of the black boxes defined by Millstein and Warshall into the setting provided by PCP, especially the File Package and the Executive Package. 5f1

[SRI-ARC]<NLS>BOXES.TXT 5f1a

Version 2 of NSW protocols

(J24591) 23-NOV-74 16:30;;;; Title: Author(s): Jonathan B.
Postel/JBP; Distribution: /NSW([INFO-ONLY]) SRI-ARC([INFO-ONLY])
; Sub-Collections: SRI-ARC NSW; Clerk: JBP; Origin: < POSTEL,
NSW-COVER.NLS;5, >, 23-NOV-74 16:26 JBP ;;;;####;

RLL 24-NOV-74 17:57 24592

Documentation Weekly response - Perhaps a monthly.

Re: (24572,) by DVN

Documentation Weekly response - Perhaps a monthly.

Perhaps Biweekly would be a better time span for the Documentation weekly (biweekly). I wish it good luck. The history of such documentation reports has been rather filled by inactive reports. I would even consideer monthly since it might be less of a burden if done that why and less of a burden on the reader to see what is happpening in documentation. 1

Documentation Weekly response - Perhaps a monthly.

(J24592) 24-NOV-74 17:57;;;; Title: Author(s): Robert N. Lieberman/RLL; Distribution: /DIRT([ACTION]) ; Keywords: Documentation; Sub-Collections: SRI-ARC DIRT; Clerk: RLL;

Report on Documentation Progress

It seems like it might also be a good idea to send a copy of any progress on documents to be used by OFFICE-1 users to KWAC for one thing to let the architects know what all is being done for them that they often times don't realize. It might demonstrate a little better what all their money is going for... 1

Report on Documentation Progress

(J24593)   25-NOV-74 07:36;;;;   Title: Author(s): Susan R. Lee/SRL; Distribution: /SRI-ARC([ACTION]) ; Sub-Collections: SRI-ARC; Clerk: SRL;

RLL 25-NOV-74 08:39 24594

Visit to NSRDC on 15 Nov 74 by RLL

This is a contact report.

Visit to NSRDC on 15 Nov 74 by RLL

(DATE) 15 Nov 74 1

(BY) Lieberman 2

(ATTENDEES) 3

Robert Lieberman (RLL) - SRI-ARC 3a

Thomas Rhodes (TRR) - NSRDC 3b

Frank Brignoli (FGB) - NSRDC 3c

Herb Ernst (HME) - NSRDC 3d

(MEDIUM) FACE-TO-FACE 4

(WHERE) NSRDC, Carderock, Maryland 5

(ACTION-ITEMS) none 6

(DISTRIBUTION) DCE JCN RLL 7

(REMARKS) 8

In talking with Frank Brignoli, Tom Rhodes, and Herb Ernst I learned several items pertinent to NSRDC, NAVSEC, Graphics, Data Management, and Navy Networking. The following are my notes from these informal discussions with my former NSRDCers. 8a

NAVSEC 8b

Pete Bono of NAVSEC is now mostly working on the Comrade project (a data management system, query langauge, file maintenance system for ship design engineers). He will return to NLS after December 1974. No one else at NAVSEC is using or near use of NLS. 8b1

Graphics 8c

Graphics at NSRDC seems to be rather inactive. The opinion is that it is too costly and funding, therefore, has fallen off as well as the interest at the management level. Even the anticipated use of the GT-40 (an intelligent graphics terminal) for graphics has been scraped. It is being used for data analysis. 8c1

NLS Applications 8d

Basically NSRDC is using NLS for several reports and for project coordination. They seem to be exploring some new ways of coordination among a geograhically distributed community. Just how sophisticated these techniques are I did not find out. 8d1

They expressed desire for the Line Processor as soon as possible since they can immediately use it with a voice grade 2400 baud modem (Valdec, I think) on a dial-up phone. 8d2

NSRDC projects 8e

Data Management Engineering 8e1

This is a 3 to 5 year project now in its first year. 8e1a

The objective is to come up with ways of classifying, organizing, engineering, and developing DMS. 8e1b

Also they are interested in written procedures to develop, establish, and use DMS. 8e1c

In addition, compiler-compiler methods and the like are being studied. 8e1d

Computer distributing problems will be looked at, hence networking, communications, large data stores, etc. are other possible avenues in this project. 8e1e

Comrade 8e2

In the Comrade project some work is being done on the DMS interface for the ship design community. 8e2a

Networking 8e3

The Navy Networking project is full speed ahead with the plan to connect all the Navy Labs. 8e3a

(DOCUMENTS) Hard copy given and received 9

(GIVEN) none 9a

(RECEIVED) none 9b

Visit to NSRDC on 15 Nov 74 by RLL

(J24594) 25-NOV-74 08:39;;;; Title: Author(s): Robert N. Lieberman/RLL; Distribution: /DCE([INFO-ONLY]) JCN([INFO-ONLY]) ; Keywords: NSRDC NAVSEC graphics Data Management Networking Marketing; Sub-Collections: SRI-ARC; Clerk: RLL;

Meeting at ONR with NSA, ONR on 7 Nov 74

This a contact report,

Meeting at ONR with NSA, ONR on 7 Nov 74

(DATE) 7 NOV 74 1

(BY) Lieberman 2

(ATTENDEES) Name of attendee (Idnum) - Organization acronym 3

Douglas Engelbart (SRI) 3a

Robert Lieberman (SRI) 3b

Susan Lee (SRI) 3c

Dennis L, Mumaugh (SRI) 3d

David R, Smith (NSA) 3e

Jim Popa (NSA) 3f

Randy Simpson (ONR Code 431B) 3g

(MEDIUM) FACE-TO-FACE 4

(WHERE) ONR small conference room, Arlington, VA 5

(ACTION-ITEMS) 6

Meeting of SRI-ARC TENEX/NLS expert with NSA and DEC people 6a

(DISTRIBUTION) DCE JCN RLL 7

(REMARKS) 8

We sat around and talked mainly with Dennis Mumaugh on the potential of using NLS at NSA, 8a

In particular with the new PDP1080,(super fast CPU and several goodies) They (NSA) are talking with DEC to procure a Tenex like operating system, 8b

It seemed that the main reason for having the new machine would be to put on NLS users, Thus it was important to know what dependence on the Tenex operating system NLS had, We agreed that someone from DEC, someone from NSA and someone from SRI-ARC should meet and discuss these technical interfaces, 8c

In summary it seems that NSA is truely hot on using NLS on one of their own machines. Their plans sound as if many people will be involved in this community of users, 8d

(DOCUMENTS) Hard copy given and received 9

(GIVEN) none 9a

(RECEIVED) none 9b

Meeting at ONR with NSA, ONR on 7 Nov 74

(J24595) 25-NOV-74 08:50;;;; Title: Author(s): Robert N. Lieberman/RLL; Distribution: /JCN([INFO-ONLY]) DCE([INFO-ONLY]) ; Keywords: NSA marketing; Sub-Collections: SRI-ARC; Clerk: RLL;

Note on Computation for ARC NSW Development

The status of things relevant to obtaining computation and net access for ARC NSW developments is the following: I will be gone on a trip starting Sun and would appreciate it if Martin could followup on the items indicated below. 1

1) Bill Carlson informs me that he has told RML to rush an order for the needed distant IMP side interfaces. Martin should followup on this with RML. 2

2) The NBS loaner IMP interface was supposed to be shipped last week. Martin should followup on this also. 3

3) Ill finally has all the parts for the other IMP interface and it should be delivered this week. Martin should followup on this also as well as getting it purchased on Capital equipment as Cox agreed some months ago. 4

4) Martin through Tom Little should check with DEC about how delivery plans on coming for the other DEC equipment PDP 11 printer etc we have on order. 5

5) I do not know what the status of our terminal orders is. Martin could you check? 6

6) Ed and Jake have been doing the internal wiring needed to hook terminal to the 11 and assume that is all cool. 7

7) I do not know what the status of Line Processors is. 8

8) The Status of PDP Tenex power is not completely clear and know that Jim is actively pursueing things with Tymshare for a second ten there and with ARPA about possibly keeping our machine a little longer if needed. I have also talked with BBN and asked Ted Strollo to send an official quote for service from them. 9

Ted thinks that about three months will be the minimum. He says if all the people who have been talking with him buy what they say they need, there will be more than 100% sold. He will reserve time first purchase order order in the door. 9a

BBN is not 100% certain their new machine will be up solid by Jan 1 but all the pieces are almost there. 9b

Bob Millstein will buy about 15-20% there and about 8-10% from us if we can give him a solid date and quote. Bill Carlson would share that percent with us and other NSW users with MCA holding option to use it if they get desparate. Bob ran some experiments with the new 132 scheduler and pie slicer using editing (SOS?), BCPL compiles loads and runs and found that about 5% was the lower bound per user for satisfactory service, Less was unsatisfactory and he did not try more, The 5% was adequate no matter what else was going on outside their piece of the pie no matter how many other bad thing they loaded in, When they ran heavy compute bound things in their piece of the pie (15%) there was degradation, 9c

At the end of this week I would like to review where the PDP 10 negotiations are at and see if we should not proceed to order 30-35% for three months from BBN, to cover any hole tht might develop until Tymshae can get a second machine up running Tenex, 9d

Note on Computation for ARC NSW Development

(J24596) 25-NOV-74 10:14;;;; Title: Author(s): Richard W. Watson/RWW; Distribution: /MEH([ACTION]) JCN([ACTION]) NPG([INFO-ONLY]) JBP([INFO-ONLY]) DCE([INFO-ONLY]) DVN([INFO-ONLY]) POOH([INFO-ONLY]) KIRK([INFO-ONLY]) ; Sub-Collections: SRI-ARC NPG; Clerk: RWW;

RWW 25-NOV-74 10:26 24597

Important NLS and Journal Demo on Dec 6 from MIT

Friday Dec 6 I will be giving an important NLS and Journal demo from an IMLAC at MIT to Licklider, Kahn, the APA Message Service committee, and some DoD people from around 6:00 to 9:00 Pacific Time (local here). The following things need doing.
1) I need to get checked out on the IMLAC.
2) We need to double check with MIT that they have the latest IMLAC program and can run it to do all normal NLS functions. Bob Belleville and I should get on the phone and double check this.
3) Jeff and Dave should be sure we have uptodate Journal indices and that they ae online.
4) I should probably have a backup account at Office 1 to use.
5) There should be someone knowledgable about things around hee at the time to deal with questions and problems with IMLAC, Tenex, NLS. Last time in Washington the demo was a mess and I would like to avoid the problems this time. Thanks Dick 1

Important NLS and Journal Demo on Dec 6 from MIT

(J24597) 25-NOV-74 10:26;;;; Title: Author(s): Richard W. Watson/RWW; Distribution: /RLB2([ACTION]) EKM([ACTION]) JCP([ACTION]) DSM([ACTION]) DCE([INFO-ONLY]) JCN([INFO-ONLY]) CHI([INFO-ONLY]) JDH([INFO-ONLY]) ; Sub-Collections: SRI-ARC; Clerk: RWW;