October 1990
Order Number: 312018

iPSC®/2 and iPSC®/860
RELEASE 3.2 SOFTWARE
PRODUCT RELEASE NOTES

intel® Corporation

REV.	REVISION HISTORY	DATE
---	Original Issue	10/90

RESTRICTED RIGHTS

Use, duplication or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the rights in Technical Data and Computer Software clause at 52.227-7013. Intel Corporation, 3065 Bowers Avenue, Santa Clara, California 95051.

PREFACE

These release notes provide the latest information on features, limitations, workarounds, and installation for the Release 3.2 system software for the following iSC products: iPSC®/2, iPSC®/2S, iPSC®/860, iPSC®/860S, and iPSC®/860Plus.

NOTE

In the remainder of the manual, we will use the term "iPSC system(s)" to refer to these products.

These notes assume that you are an application programmer, familiar with the C or Fortran language and the UNIX operating system.

For more information, refer to the Release 3.2 system software customer letter that accompanied your software.

For installation instructions, refer to Chapter 4.

ORGANIZATION

Chapter 1	Describes features of the Release 3.2 system software.
Chapter 2	Describes known limitations and their workarounds.
Chapter 3	Describes information necessary to convert PRM assembly code to ABI assembly code.
Chapter 4	Provides system software installation information.

APPLICABLE DOCUMENTS

For more information, refer to these iPSC, Intel and other manuals:

iPSC® System Manuals

iPSC®/2 and iPSC®/860 C Language Reference Manual
Describes the C compiler for the iPSC system.

iPSC®/2 and iPSC®/860 Fortran Language Reference Manual
Describes the Fortran compiler for the iPSC system.

iPSC®/2 and iPSC®/860 Math Libraries Reference Manual
Describes the math libraries available on the iPSC system.

iPSC®/2 and iPSC®/860 Programmer's Reference Manual
Describes iPSC system commands and system calls (both C and Fortran).

iPSC®/2 and iPSC®/860 System Acceptance Test User's Guide
Tells how to use the System Acceptance Test.

iPSC®/2 and iPSC®/860 System Administrator's Guide
Describes the system administration tasks related to operating and maintaining an iPSC system.

iPSC®/2 and iPSC®/860 User's Guide
Overviews the iPSC system, including hardware and software architectures. Tells how to develop and run programs.

iPSC®/2 Ada Program Development Guide
Describes and tells how to use the tools for developing Ada programs for the iPSC/2.

iPSC®/2 Ada Programmer's Reference Manual
Describes all Ada routines and commands for the iPSC/2 system.

iPSC®/2 DECON User's Guide
Tells how to use DECON, the iPSC/2 concurrent debugger.

iPSC®/2 Lisp Language Reference Manual
Describes the Lisp implementation and its extensions that run on the iPSC nodes.

iPSC®/2 Lisp Programmer's Reference Manual
Describes the iPSC/2 Lisp user-interface and the iPSC/2 Lisp concurrent constructs.

iPSC®/2 Simulator Manual
Tells how to use the iPSC/2 Simulator for software development.

iPSC®/2 VME Interface Reference Manual
Describes the installation and development of software drivers for the VME Interface Adapter board.

Intel® Manuals

Intel® NFS for System V/386 Programmer's Guide and Reference
Describes the NFS programming environment and tools.

Intel® NFS for System V/386 User's/System Administrator's Guide and Reference
Describes the NFS programming environment and provides user and system administration information.

Intel® TCP/IP for SYSTEM V/386 Administrator's Guide and Reference
Describes TCP/IP Network administration.

Intel® TCP/IP for SYSTEM V/386 Programmer's Guide and Reference Manual
Describes the TCP/IP Network programming environment and provides information on programming tools.

Intel® TCP/IP for SYSTEM V/386 User's Guide and Reference
Describes the TCP/IP Network programming environment and provides user information.

i860™ 64-Bit Microprocessor Assembler and Linker Reference Manual
Tells how to use the i860 assembler and linker.

i860™ 64-Bit Microprocessor Programmer's Reference Manual
Tells how to use the i860 microprocessor.

i860™ Microprocessor Vector Primitive Library Reference Manual
Provides a complete list of the vector primitives for VAST2 for RX nodes.

SYP301 Installation and User's Guide
Tells how to install and start the System Resource Manager. Also provides hardware technical data.

Other Manuals

C: A Reference Manual - Harbison and Steele
Describes the C programming language.

Common Lisp: The Language - Guy L. Steele Jr.
Describes the Lisp programming language.

Reference Manual For The Ada Programming Language - ANSI/MIL-STD-1815A-1983
Describes the Ada programming language.

The C Programming Language - Kernighan and Ritchie
Describes the C programming language.

Volume 0, Protocol Reference Manual - O'Reilly & Associates
Provides reference information on X Network Protocol, the language for communication between the X server and the X client.

Volume 1, Xlib Programming Manual - O'Reilly & Associates
Tells how to program using the X library, the lowest level programming interface to the X window system.

Volume 2, Xlib Reference Manual - O'Reilly & Associates
Provides reference information for programming using the X library.

Volume 4, X Toolkit Intrinsics Programming Manual - O'Reilly & Associates
Tells how to program using the X Toolkit.

Volume 5, X Toolkit Intrinsics Reference Manual - O'Reilly & Associates
Provides reference information for programming using the X Toolkit.

UNIX System V Manual Set
Describes UNIX System V.

NOTATIONAL CONVENTIONS

This section describes the following notational conventions:

- Type style usage
- Examples in text

Type Style Usage

This manual uses the following type style conventions:

- **Bold** type style is used for anything that must be entered exactly as shown, including:

  Command names (including absolute pathname versions)
  Switches, flags, and options
  System Call names
  Routine names (predefined *and* user-defined)
  Reserved words

- *Italic* type style is used for:

  Variables
  File names (including absolute pathname versions)
  Directory names
  Process names
  User names
  Emphasis

- ***Bold-italic*** type style is used for user input described in text (what you enter in response to some prompt).

- `Plain-Monospace` type style is used for:

  Computer output (prompts and messages)
  Code
  Error messages
  Values of variables

- ***`Bold-Italic-Monospace`*** type style is used for user input displayed in an example (what you enter in response to some prompt)

- **Bold-Monospace** type style is used for the names of keyboard keys (which are also enclosed in angle brackets). For example:

<Alt>	**<Backspace>**
<Break>	**<Ctrl>**
<Delete> or **<Del>**	**<Enter>**
<Esc>	**<Tab>**

For a key that prints, the result of pressing the key is used. For example, **s** means press the key labeled **<S>**, and **S** means press and hold the **<Shift>** key while pressing the **<S>** key.

A dash indicates that the key preceding the dash is to be held down *while* the key following the dash is pressed. This has the effect of producing a single keystroke (which is why there is only one set of angle brackets). For example:

<Ctrl-S>
<Ctrl-@>
<Ctrl-Alt-Del>

Note that there are several ways to reference some keys. For example, **S** and **<Shift-s>** mean the same thing. In such cases, the simplest method (**S**) is usually used.

Examples in Text

This manual uses `monospace` type style for examples.

In examples of interactive sessions, user input appears in ***bold-italic-monospace*** type style to distinguish it from computer output. For example:

```
# getcube -t20                                          (user input)
getcube successful: cube type 2m8n0 allocated          (computer output)
```

Many examples contain annotations that describe specific parts of the example. These annotations (which are not part of the example code or session) appear in *italic* type style.

TABLE OF CONTENTS

CHAPTER 1
PRODUCT FEATURES

CHAPTER 2
LIMITATIONS AND WORKAROUNDS

CHAPTER 3
THE APPLICATION BINARY INTERFACE

CHAPTER 4
SOFTWARE INSTALLATION

LIST OF TABLES

PRODUCT FEATURES 1

INTRODUCTION

The R3.2 system software runs on iPSC®/2 (with 386™-microprocessor-based CX nodes) and iPSC®/860 systems (with RX nodes based on the i860™ microprocessor or with a combination of CX and RX nodes).

NOTE

In the remainder of the manual, we will use the term "iPSC system(s)" to refer collectively to the iPSC®/860, iPSC®/860S, and iPSC®/860Plus products.

In general, the iPSC/860 system operates the same as the iPSC/2 system does, using the same commands and calls, but with added computational speed. Table 1-1 shows the major operational differences between the RX nodes and the CX nodes.

Table 1-1. Major Differences in the Operation of RX Nodes and CX Nodes

RX Nodes	CX Nodes
Peak performance of a 40-MHz i860 processor rated at 80 MFLOPS for single precision, 40 MFLOPS for double precision, and at 27 VAX MIPS	Peak performance of the processor rated at approximately 0.3 MFLOPS, and two or three VAX MIPS
Single process allowed on nodes, only pid 0 allowed	Multiple processes allowed on nodes
8M bytes of memory per node	4 to 16M bytes of memory per node
Fortran and C compilers available	Fortran, C, Ada, and Lisp compilers and Lisp interpreter available
On-board fast vector processing	Vector processing enhancement available with the VX option
On-board fast scalar processing	Scalar processing enhancement available with the SX option
Vectorizing preprocessor not available	VAST2 vectorizing preprocessor available
Node programs must be compiled and linked with the **-i860** switch, while host programs require 386 compilation and links	Node and host both 386-based

NEW FEATURES

The iPSC Release 3.2 system software provides the following new features:

1. **NX/2 operating system on RX node**
   NX/2 has been ported to run on i860-microprocessor-based RX nodes. It includes enhancements to reliability, performance, and some new features. Reliability enhancements consist of bug fixes and some redesign for improved protection of system data structures. Message passing performance has also been improved.

   Added features include enhancement of **getcube** to enable specification of CX or RX nodes when you allocate a cube and the addition of two new clock calls: **dclock()** and **hwclock()**.

2. **Improved i860™ C and Fortran compilers**
   The i860 C and Fortran compilers have been improved over those delivered with the Release 3.2 preproduction software. New **cc** and **f77** command line switches have been added to make it easier to compile and link 386 and i860 programs.

3. **Faster linker for i860™ programs**
   Linker speed for RX programs has been greatly improved over that delivered with the Release 3.2 preproduction software.

4. **IEEE arithmetic for RX programs**
   This release supports IEEE divide, square root, and unordered comparison operations for RX programs.

5. **DECON on RX nodes**
   The DECON concurrent debugger has been extended to provide the same capabilities for RX nodes that it does for CX nodes.

LIMITATIONS AND WORKAROUNDS 2

This chapter describes known limitations and suggested workarounds for the following Release 3.2 system software components:

Documentation
NX/2 node executive for the CX and RX nodes (referred to collectively as "NX")
RX node mathematical operations
Node shell (**nsh**)
Concurrent File System
Concurrent File System tape
General Concurrent Workbench functions
Concurrent Workbench assembler and linker
Concurrent Workbench C
Concurrent Workbench Fortran
Concurrent Workbench remote host
DECON concurrent debugger
UNIX software
TCP/IP
CDP

NOTE

Read the following sections carefully. Report any problems you encounter while using your iPSC system to iSC Customer Support at:

1-800-421-2328 (Customer Support Hotline)
(44) 793 641 469 (in England)
Your Local Intel Sales Office (in Europe)
support@isc.intel.com (Internet address)

DOCUMENTATION

Four Intel i860 manuals are shipped as part of the iPSC documentation package. These four manuals are shrink-wrapped in a single package. Only two of these manuals apply to the iPSC/860 systems. The two manuals that apply are the *i860™ 64-Bit Microprocessor Assembler and Linker Reference Manual* and the *i860™ 64-Bit Microprocessor Simulator and Debugger Reference Manual.*

There are two manuals in the package that are not supported and should not be used:

- *i860™ 64-Bit Microprocessor Math Library Reference Manual*
  The iPSC system does not support the i860 math libraries as documented in this manual. The iPSC supports the math libraries described in the UNIX System V/386 documentation.

- *i860™ 64-Bit Microprocessor Object File Utilities Reference Manual.* The object file utilities described in this manual have much the same functions as those that the iPSC supports, but the names are different. iPSC object file utilities are described in the *iPSC®/2 and iPSC®/860 User's Guide.*

NX NODE EXECUTIVE

1. **Use of a node pid number other than 0** ***(RX nodes only)***
   If you send a message to an RX node using a pid (process id) other than 0, the message will be delivered to pid 0 and no error message will be returned.

2. **Message length**
   Check that the *len* parameter in a node send or receive call does not exceed the size of the buffer. Unpredictable results can occur if *len* exceeds the buffer size. (Refer to the *iPSC®/2 and iPSC®/860 Programmer's Reference Manual* for more information on these parameters.)

3. **Invalid buffer pointers**
   Invalid buffer pointers in C node programs will sometimes be accepted as valid and not return an error message.

4. **Node may deadlock with file I/O**
   If a node's message buffers are filled but no receives are posted, attempts to do file I/O may fail. To avoid this failure, you must make sure that the node receives pending messages.

5. **flushmsg() may not flush all messages**
   The **flushmsg()** routine may not flush a message that is in transit.

6. **cubeinfo command displays only system name**
   The **cubeinfo()** routine returns the system and domain name for the *srmname* and *hostname* fields. The **cubeinfo** command truncates the domain name, and displays just the system name.

7. **Prints originating within a handler**
   Prints originating from within a handler routine may overwrite or merge with the prints from other nodes.

8. **load() error message misleading**
   If you try to load a nonexistent file, the wrong type of file (386 vs. i860 file), or a null-length file, you will get the following error message:

   ```
   Exec format error
   ```

9. **Sending long messages to RX nodes**
   If you send a long message to an RX node other than yourself, the send will not return until the receiving process has been loaded.

   Differences between RX and CX nodes when using hrecv()
   On RX nodes, using **hrecv()** to receive a message that is too long to fit in the buffer causes the program to terminate with an error message. On CX nodes, the receive completes, the message is lost, and no error is returned.

10. **A fork() failure on an RX node can cause subsequent fork()s to fail**
    If a **fork()** on an RX node fails, subsequent **fork()**s may also fail until you do a **killcube**.

11. **Not all messages are flushed when a node process kills itself**
    When a node process calls **killcube()** such that it kills itself (for example, **killcube(-1,-1)**), pending messages for that node are not flushed. You must subsequently use the **killcube** command to clean up those messages.

12. **Calling sleep() and hsend(), hrecv(), or hsendrecv() in a node program**
    Calling **sleep()** and **hsend()**, **hrecv()**, or **hsendrecv()** in a node program may cause the program to fail to return from the handler routine.

    Workaround: Don't call **sleep()**. Instead, you can use **mclock()** in a loop.

13. **pipe(), fork(), and exec() calls not fully supported**
    UNIX-compatible calls **pipe()**, **fork()**, the **exec()** family, **setuid()**, and **setgid()** for the nodes are only partially implemented. They are used by the node shell (**nsh**) and may be useful in porting programs from the UNIX environment to run under **nsh**. Using them in node programs is *strongly* discouraged.

14. **Change in buffering long messages on RX nodes**
    NX uses a three trip protocol to send long messages (over 100 bytes), to ensure that the receiving node has room to store the message before it is sent. When a long message probe is received and enough memory is available, NX/2 will immediately allocate a buffer and receive the message. However, NX/2 on the RX node will wait until the node process has no more computations to do before allocating the buffer and receiving the message. This has little effect on the receiving node and may improve performance by giving the receiving node time to post the receive. However, it will seriously slow down the sending node if it uses **csend** to send a long message to a node that is busy computing.

For example, on a two node iPSC/2 system, the following code will overlap the computing almost completely. On the iPSC/860 system the computing will be serialized because each **csend** will hang until the other node finishes computing and posts the receive.

```
      real outbuf(200), inbuf(200)

      if(mynode() .eq. 0) then
         do 10 i = 1, 1000
           call csend(0, outbuf, 800, 1, 0)
                .
                .
                .
           COMPUTE
                .
                .
                .
           call crecv(0, inbuf, 800)
10       continue
      else
         do 20 i = 1, 1000
                .
                .
                .
           COMPUTE
                .
                .
                .
           call crecv(0, inbuf, 800)
           call csend(0, outbuf, 800, 0, 0)
20       continue
      endif
      end
```

The solution is to use either **isend()**s or **irecv()**s to keep the process from blocking. For example, the following code accomplishes the same result and does not suffer a performance degradation on the iPSC/860 system.

```
      real outbuf(200), inbuf(200)

      if(mynode() .eq. 0) then
         do 10 i = 1, 1000
           call csend(0, outbuf, 800, 1, 0)
           it = irecv(0, inbuf, 800)
                .
                .
                .
```

```
                COMPUTE
                     .
                     .
                     .
                call msgwait(it)
10           continue
          else
             do 20 i = 1, 1000
                     .
                     .
                     .
                COMPUTE
                     .
                     .
                     .
                if(i .ne. 1) call msgwait(it)
                call crecv(0, inbuf, 800)
                it = isend(0, outbuf, 800, 0, 0)
20           continue
             call msgwait(it)
          endif
          end
```

In some applications, it may be necessary to declared an extra buffer to use in the **isend** or **irecv**, but the memory must be available if the system was going to buffer the message.

RX NODE MATHEMATICAL OPERATIONS

RX multiplication of denormals
There is a problem in the i860 microprocessor's IEEE-754 floating point exception handler when multiplying denormal numbers. For pipelined and scalar multiplication, wrong answers may result. Also, for scalar multiplication, a floating point register may be corrupted. Which register is corrupted depends upon the previous floating point instructions.

Compiling without optimization may help.

MATH LIBRARIES

The vector library for the iPSC/860 system is included in the software to provide the compatibility migration path from the 386 vector product to the i860 microprocessor. These libraries are not currently optimized to run at their peak speed on the i860 microprocessor.

NODE SHELL

1. **On an RX cube, at least four nodes are required to run the node shell *(RX nodes only)***
   To run the node shell (**nsh**), make sure that you get at least four RX nodes. The node shell can be run on one CX node. Also, you cannot use pipes (|) in the node shell on a single RX node. RX nodes run one process per node. The node shell requires an RX cube of at least four RX nodes. To run pipes in the node shell, you must have additional nodes for each of the pipe processes.

2. **No error message when wrong type of executable is called**
   The node shell (**nsh**) returns no error messages when an executable of the wrong type is called (for example, a 386 executable file in an i860 cube).

3. **nsh wildcard (*) does not work properly for filenames larger than 14 characters**
   The **nsh** wildcard character (*) does not expand properly for CFS filenames longer than 14 characters.

4. **Changing directory names**
   The **mv** command does not work on directories, even to change the name of the directory. The only way to do this is to copy the whole directory structure to the new name and remove the old structure.

5. **Loading and killing processes on nodes not assigned to you**
   Under certain circumstances, it is possible to load and kill processes on nodes that are not assigned to you. However, this will not occur if you use the system as documented.

6. **-c option not available for nsh**
   The **-c** option of the **nsh** command is not supported.

7. **Filling the file system using lsize**
   When you fill the CFS file system using the **lsize** command in the **nsh** shell, you receive the following error messages

   ```
   lsize: cannot allocate filename
   lsize: Invalid argument
   ```

   Instead, you should receive the following error message:

   ```
   No space left on device
   ```

8. **RX version of nsh does not handle interrupts correctly**
   The RX version of **nsh** does not handle interrupts correctly. It ignores a single interrupt. Two interrupts abort the current command and/or **nsh** itself. Multiple interrupts abort **nsh**.

9. **Simultaneous tar commands (backgrounded) hang nsh**
   Simultaneous **tar** commands in the background can hang **nsh**. It is necessary to **reboot** if this occurs.

CONCURRENT FILE SYSTEM™

1. **Killing an application that is allocating a large amount of space to a file will hang CFS**
   Using **relcube**, **killcube**, or hitting the **<Del>** key in **nsh** while an application is allocating a large amount of space to a file will hang CFS. One of the following two messages is added to the log file when this happens:

   ```
   Diskproc internal error: No fcaux in ALLOCATE
   ```

   or

   ```
   Diskproc internal error: Not open in ALLOCATE
   ```

   If this should happen, call iSC Customer Support.

2. **The load command run on the SRM silently fails to load executables that reside on CFS**
   The **load** command run on the SRM silently fails to load executables that reside on CFS.

   Workaround: Load CFS programs under **nsh**.

3. **Attempting to create a large file in CFS may corrupt CFS**
   Attempting to create a file that is greater than 2G bytes in CFS may corrupt CFS.

4. **Cannot use open() on a directory name**
   You cannot use **open()** on a directory name in CFS. Use **opendir()** instead.

5. **umask() works only on CFS**
   The **umask()** routine on the nodes works only on CFS. It is not supported for SRM files.

6. **mkdev gives incorrect error message**
   If **mkdev** is used to create a special file that already exists, it prints the error message

   ```
   Not a directory
   ```

   rather than the expected message

   ```
   File exists
   ```

7. **Cannot modify permissions or owner of devices created by mkdev**
   There is no way to change the permissions or owner of links that are created by **mkdev**. This prevents giving restricted access to some devices.

8. **cbackup and crestore not available on SRM**
   cbackup and **crestore** are no longer available on the SRM. Use the UNIX **backup** and **restore** commands to backup and restore SRM files. Use the **nsh cbackup** and **crestore** commands for CFS files.

9. **Changing disk configuration between a cbackup and crestore**
   If a disk is added between a full **cbackup** and **crestore**, the **crestore** damages the file system, but the damage isn't visible until the next **bootcube**.

10. **Execution of cbackup and crestore not properly restricted**
    Because the **cbackup** and **crestore** commands can change the entire file system, only the superuser should be able to use them. However, the commands do not currently enforce this restriction.

11. **iowait() returns incorrect error message**
    If an error occurs when you are using **iread()** (such as reading past EOF), **iowait()** may print:

    ```
    Error 0
    ```

12. **lsize() may set errno incorrectly**
    lsize() sets *errno* to an incorrect value when it encounters an `insufficient space` error or an invalid `whence` value error.

13. **File date information not updated until file is closed**
    The date indicating when a file was last modified is not updated until the file is closed. Therefore, checking this date on a file that is currently open and being written to produces misleading information.

14. **Invalid file descriptor argument causes incorrect error message**
    If you use an invalid file descriptor as an argument to **setiomode()**, the following error message is printed:

    ```
    Not supported on SRM
    ```

    The error should be

    ```
    Bad file number
    ```

15. **Under certain conditions, nsh refuses commands**
    nsh will occasionally refuse commands and return the following error message:

    ```
    No more processes
    ```

    Workaround: Use **killcube**, and, if necessary, use **bootcube** to clean things up and recover.

16. **Incorrect restrictvol() error message**
When you give **restrictvol()** an invalid file descriptor, it incorrectly displays the following error message:

```
Not supported on the SRM
```

It should display the following:

```
Bad file number
```

17. **File system permissions are not always handled correctly**
In some cases, file system permissions are not handled correctly. In particular, you can not remove the files of others from your directory.

CONCURRENT FILE SYSTEM™ TAPE

1. **Tape device file must exist before using CFS tape drive**
Refer to the **mkdev** command in the *iPSC®/2 and iPSC®/860 Programmer's Reference Manual.*

2. **End of medium detection**
End of medium detection works reliably only for writes to tape. When the end of medium is reached, the **write()** returns a `-1` and sets *errno* to `ENOSPC`. The application must explicitly close the device and then reopen it after a new tape has been mounted.

3. **CFS tape device file entry date is not updated**
The date field in a CFS tape device file entry does not reflect the last write time.

4. **Multiple names for a tape drive treated the same**
If there is more than one name for a tape drive (for example the Volume 1 drive is named *tape1* and *tape2*) and one of the names is set to `no rewind`, both names will be set to `no rewind`.

5. **Invalid tape operations may hang tape driver**
Following a **write()**, any tape forward movement that is not a **write()** should return an error, but does not. If this occurs and you do a **rewind()**, the next forward movement may cause the tape driver to hang. To recover, use **bootcube**.

6. **Attempting to write on a write-protected tape**
Writing to a write-protected tape does not return an error or write to the tape. A second attempt will hang the tape driver. To recover, use **bootcube**.

7. **Reading a tape using the wrong block mode hangs the tape driver**
Reading a tape in a fixed block mode that is different than the one in which it was created hangs the tape driver. To recover, use **bootcube**.

8. **Accessing an off-line tape drive**
   Attempts to access a tape drive that is off-line cause CFS to hang. To recover, use **bootcube.**

9. **Must read same size blocks as were written**
   Using **cread()** to read blocks on a tape device that are larger than those written results in the message,

   ```
   Attempt to read past end of file
   ```

   even when this is not the case.

10. **Using crestore with the CFS tape device in variable block mode**
    Using the node **crestore** command with the CFS tape device in variable block mode damages the CFS file system.

    Workaround: Use the **cbackup** and **crestore** commands with the CFS tape device set to a fixed block mode.

11. **Backing off from the beginning of a tape hangs the driver**
    Using a backspace file (**MTBSF**) or backspace record (**MTBSR**) command when the tape is at the beginning of the tape hangs the tape driver. To recover, use **bootcube.**

12. ***diskproc* may hang**
    Occasionally, the *diskproc* may hang and write either of the following messages in */usr/ipsc/log/iocube.log*:

    ```
    ESP error status
    ```

    or

    ```
    ESP FIFO not clear in MSGIW
    ```

    This is a fatal error for which there is no workaround. To recover, use **bootcube.**

13. **Asking for unsupported tapemode hangs CFS**
    If you request an STK nine-track tape drive to change to 800 bpi density (only 1600 and 6250 are supported), the **tapemode** command hangs. Two interrupts are required to break out of the hang, but **nsh** dies in the process. The cube must be rebooted to return the system to normal operation.

CONCURRENT WORKBENCH™

1. **waitcube and killcube**
   The **waitcube** and **killcube** commands occasionally fail to return. To recover, press the **<Del>** key to kill the command, use ***kill -9*** to clean up unwanted *fservers*, and then do a **relcube**. If the **relcube** fails to return, press the **<Del>** key, then do a **bootcube**.

2. **Host pid 0**
   If there is a host program in the background that has done a **setpid(0)**, some cube commands will die early with the following message

   ```
   (host) ___: Pid already in use
   ```

   This is because **waitcube, startcube, load,** and **killcube** try to use **setpid(0)**.

3. **Nodes marked unusable**
   If you get a message saying,

   ```
   nn node(s) not responding
   ```

   (where *nn* is the number of nodes), do a **bootcube** to reset the nodes. If problems persist, use **cdp** to check for bad node boards. The *iPSC®/2 and iPSC®860 System Administrator's Guide* describes **cdp**.

4. **Incorrect cubeconf settings can cause red LEDs to flash continuously**
   If *cubeconf* slot fields are set to EMPTY but contain 386 nodes, the red LEDs flash continuously after a **bootcube**. See the *iPSC®/2 and iPSC®860 System Administrator's Guide* for more information on *cubeconf*.

5. **Message length**
   Check that the *len* parameter in a host send or receive call does not exceed the size of the buffer. Unpredictable results can occur if *len* exceeds the buffer size.

6. **Invalid buffer pointers**
   Invalid buffer pointers in C host programs will sometimes be accepted as valid and not return an error message.

7. **Host long messages**
   Host programs cannot use **csend()** or **isend()** followed by **crecv()** to send long messages (greater than 100 bytes) to themselves. Also, a node program sending a message of greater than 100 bytes to the host will not complete the send operation until the host application posts a receive for the message.

   Workaround: Use **irecv()**, **csend()**, **msgdone()**, or **msgwait()**.

   The following example shows a code fragment from a host program:

```
id = irecv ( 1, buf, sizeof(buf));
       .
       .
       .
csend ( 1, msg, sizeof(msg), myhost(), mypid());
msgwait(id);
```

   The following host code works only if the send and receive buffers are different:

```
id = isend ( 1, msg, sizeof(msg), myhost(), mypid());
       .
       .
       .
crecv ( 1, buf, sizeof(buf));
msgwait(id);
```

8. **Receiving process must be alive at time of send for long message buffering to work**
   Long message buffering on the nodes only works if the receiving process is alive at the time of the send. On RX nodes, the message receive must also be posted.

9. **Fatal error on RX nodes when a message too long for buffer received by hrecv()**
   Using **hrecv()** to receive a message that's too long to fit in the buffer can cause a fatal error on RX nodes.

10. **Releasing a cube**
    Use **killcube** prior to releasing the cube. **relcube** alone does not always clean up processes properly, causing subsequent programs to fail. Using **killcube** avoids this problem. Use the **ps -ad** node command to verify that the clean-up is complete.

11. **setsyslog()**
    Calling **setsyslog()** when the host program is already piping the output through **syslog** causes the host program to hang. To recover, press the **<Del>** key.

12. **newserver()**
    A host program cannot call **newserver()** if its output is being piped through **syslog**. Doing so causes the host program to hang or be killed. To recover, press the **<Del>** key.

13. **Node memory may not be recovered**
    killcube and **relcube** may fail to recover node memory, resulting in the following error message:

    ```
    Not enough memory
    ```

    Use **bootcube** to recover.

14. **killcube flushes file server messages**
    If a host program calls **killcube** before nodes complete file I/O, some output may be lost.

    Workaround: Call **waitcube** before calling **killcube**.

15. **Cube allocated by bootcube**
    When **bootcube** is executed on an SRM, a zero-node cube named "iocube" is allocated. This is necessary for all systems. It reduces the number of cubes that users can allocate from 10 to 9.

16. **Maximum of 11 outstanding irecv()s may be posted on an SRM**
    A maximum of 11 outstanding **irecv()**s per cube may be posted on an SRM by a host program. If you attempt to post more than this, an error message displays, and the process aborts. The maximum for the remote host is 12.

17. **killcube hangs if load cannot complete**
    If the node message buffers are filled before a load is complete, a subsequent **killcube** hangs. This situation can be avoided by ensuring that the load is complete before any process sends a large number of messages to the nodes.

18. **Profiling host applications may cause problems**
    The **getcube()** and **newserver()** routines abort if the host application in which they are called is compiled using the profiling switch **-p**.

19. **Redirecting I/O to node programs**
    The **getcube** and **newserver** commands can no longer be used to redirect I/O to an application that is reading from *stdin*. Use the **waitcube** command instead.

20. **Usage message from load in both nsh and SRM shell incorrect**
    The usage message from the **load** command in both the **nsh** and SRM shell is incorrect; it should be as follows:

    ```
    Usage: load [-cpHDV] [node...] file [arguments...]
    ```

21. **load error message misleading**
    If you try to load a nonexistent or null-length file, or if you try to load a CX node executable on an RX node (or vice-versa), you will get the following error message:

    ```
    Exec format error
    ```

22. **Errors from getcube produced by commser not responding**
The following **getcube** error may occur, especially when someone is linking an i860 application during the **getcube**:

```
Permission denied
```

If this error does occur, wait awhile and try another **getcube**. If several **getcubes** in a row fail, you may have to use **bootcube**.

23. **Load command may not check permissions appropriately for NFS files**
If a node program is stored in a directory mounted on the SRM using NFS, and the directory path is not world executable all the way down, the **load** command returns the following error, even though the user has all the permissions on the file and the path directories:

```
Permission denied
```

CONCURRENT WORKBENCH™ ASSEMBLER AND LINKER

1. **Directive incorrectly used**
In assembling data objects declared with the **.data** instruction, the 386 assembler creates a large temporary file on the SRM. This file can exhaust all available file space on the SRM and abort the assembly.

Workaround: Such data objects will not create large files during assembly if they are declared using the equivalent **.bss** instruction.

2. **The 386 assembler is not documented**
as is documented only as **as(1)** in the UNIX System V/386 manuals. The actual assembly instructions are not documented.

CONCURRENT WORKBENCH™ C

The C compiler, gcc, is located in */usr/bin* and is linked to */usr/bin/cc*.

To invoke Green Hill's compiler, enter:

```
cc [-i860 | -i386 ]
```

or

```
gcc [-i860 | -i386 ]
```

The UNIX-provided C compiler is in */bin/pcc*.

The **-i386** switch is the default, and should be used to compile host programs that will run on the SRM. You must use the **-i860** switch to compile and link programs that will run on the RX nodes.

The following are Concurrent Workbench C limitations and workarounds:

1. **Compiling and linking i860 programs without -node switch results in an error**
   If you do not use the **-node** switch when linking programs for the RX node, you may receive the following run-time error message:

   ```
   (node n, pid 0): Invalid system call at xxxxxxxx
   ```

2. **Deeply nested macros and -ansi switch cause internal compiler error** ***(386 compiler only)***
   When macros are deeply nested (greater than 4 levels) and the undocumented and unsupported **-ansi** switch is used, the compiler quits with the following error message:

   ```
   Internal Compiler Error
   ```

3. **Optimization causes loss of `end block` records** ***(i860 compiler only)***
   When optimization and symbolic debug are both selected (**-O -g**), not all `end block` records are generated, causing the assembler to issue error messages about unmatched `begin block` records.

   Workaround: Use either optimization (**-O**) or symbolic debug (**-g**), but not both simultaneously.

4. **Bad casting practice causes bad code generation** ***(i860 compiler only)***
   Casting a byte-aligned array of **char** which is in a static structure into an **int** or **long** causes the assembler to generate errors about `misaligned references`. This is actually an assembler bug, but the compiler does not give a warning, as expected.

   Workaround: Change the **char** array into a union of **char** array and **int** or **long** as required.

5. **asm386 directive support**
   The C compiler is missing some support for the **asm386** directives.

6. **-C compiler switch**
   The **-C** compiler switch is not implemented.

7. **Illegal external variable names**
   You may not use the following names as external variables in your host programs: **align, any, C, D, E, e, f, FP, herror, host, hostbuf, HOSTDB, hostf, index, IP, KS, L, line, LogFile, LogMask, nbuf, net, P, proto, R, recv, S, s, send, shifts, syslog,** or **token.** Network library */usr/lib/libsocket.a* has defined these names to be external. Errors such as the following may occur at runtime if you use them in an application:

   ```
   Bus error - Core dump
   ```

8. **-Z618 compiler switch**
   Using the **-Z618** i860 C compiler switch may cause alignment differences between structures with the i860 compiler versus the 386 compiler.

CONCURRENT WORKBENCH™ FORTRAN

1. **Compiling and linking i860 programs without -node switch results in an error**
   If you do not use the **-node** switch when compiling and linking programs for the RX node, you may receive the following run-time error message:

   ```
   (node n, pid 0): Invalid system call at xxxxxxxx
   ```

2. **Possible loss of write to standard output**
   In the case where a member of the output list of a **write** or **print** statement (to standard output) is a reference to a function that also includes a **write** or **print** statement to standard output, the write to standard output within the function may be lost. For example, the following statement may lose a print statement within the function **foo()** so no output would appear on standard output:

   ```
   print *, foo(arg)
   ```

   The ANSI Fortran 77 standard indicates that this is an illegal use of the **write** or **print** statement.

3. **Functions that include write statements cannot be called from within a write statement**
   A Fortran **write** statement that calls a function that contains a **write** statement causes the following runtime error message:

   ```
   Bus error - core dumped
   ```

   The ANSI Fortran 77 standard indicates that this is an illegal use of the **write** statement.

4. **csendrecv(), cubeinfo(), ecmp(), ginv(), gray(), and restrictvol() return integers**
   The iPSC system functions **csendrecv()**, **cubeinfo()**, **ecmp()**, **ginv()**, **gray()**, and **restrictvol()** must be declared as integers, either with the include file */usr/include/fcube.h* or with user-provided declarations.

5. **dclock() returns a double precision number**
   The iPSC system function **dclock()** must be declared as double precision, either with the include file */usr/include/fcube.h* or with user-provided declarations.

6. **Fortran constants must be explicitly tagged as doubles to be stored as double precision**
   In compliance with the ANSI Fortran 77 standard, Fortran constants are not stored as double precision (the default for CRAY systems) unless explicitly tagged as doubles.

7. **Output lines from write or print statement**
   Instead of using the first character in a **write** or **print** statement as a printer control character, the character is printed. Also, an extra space is added for each continuation line in these statements. Use the **-vms** compiler switch to make the **write** and **print** statements use the first character as a printer control character and to eliminate the extra space on continuation lines.

8. **Error messages for the following are not provided:**

   - Extra characters on a **DO** statement
   - **DATA** statements used with a variable declared in a common block, but not in **BLOCK DATA** (VMS extensions)
   - Trigonometric function whose arguments are greater than 2**63
   - Same formal argument name used in a subroutine statement more than once
   - Logical variable used as array index
   - Integer variable used as logical (VMS extension)
   - Mix of character and any other type within a common block (VMS extension)

9. **-i2 compiler switch**
   The **-i2** compiler switch (integers default to 2 bytes) returns an incorrect error message for large integer arrays, and the compilation aborts.

10. **Invalid invocation line switches**
    Invalid compiler switches are silently ignored by the compiler.

11. **VAX/VMS extensions**
    Compiling with the **-vms** switch makes VAX/VMS Fortran extensions available during compilation. For more information, refer to the *VAX-11 Fortran User's Guide and Language Reference Manual*, available from Digital Equipment Corporation.

12. **Running out of node board memory**
    If you run out of node board memory while running a Fortran application, you may receive one or more of the following messages:

```
Fortran runtime error on external file "" (106): Buffer too large
Stack overflow
Memory fault
```

    This can happen when making the first write to a file, as the program tries to allocate a buffer for the file.

13. **I = MOD(J, 0) causes error message**
    The expression **I = MOD(J, 0)** in a host program causes a runtime divide-by-zero exception, and in an RX node program the runtime value of **I** will always be **J**. This generates illegal assembler instructions, causing the assembler to issue a warning as follows:

    ```
    WARNING: Shift count isn't in 0-31 range
    ```

14. **Formatted print statements on multiple nodes**
    Using Fortran formatted print statements that contain carriage return control characters ("/") on multiple nodes may cause prints from multiple nodes to merge together. That is, a message line from one node may have portions of a message from another node mixed in.

15. **Node and SRM versions of rewind**
    The node and SRM versions of the Fortran **rewind()** routine only reset the file pointer to the beginning of the file. They do not shrink the file, when appropriate, as does the standard **rewind()** routine.

16. **Character string padding in Fortran etos() and stoe() routines**
    The Fortran **etos()** routine pads a string with NULL characters (ASCII 0) instead of blanks (ASCII 32). Fortran **stoe()** expects strings to be padded with NULL characters, not blanks. This may cause the following unexpected behavior:

    - Using a string padded with blanks in **stoe()** may cause the following error:

      ```
      stoe: Invalid size
      ```

      Workaround: Use strings generated by **etos()** in **stoe()**.

    - String comparisons may fail. A string padded with NULLs will not match a string padded with blanks.

17. **-618 compiler switch**
    Using the **-618** i860 Fortran compiler switch may cause alignment differences between 386 microprocessor common blocks and i860 microprocessor common blocks.

CONCURRENT WORKBENCH™ REMOTE HOST

1. **load() or exec() processes executing on nodes cannot find files on remote host**
   load() and **exec()** calls executed on the nodes try to find the files on the SRM, not the remote host workstation.

   Workaround: Any files that need to be **loaded** or **execed** from the nodes should be resident in the SRM file system. Either **rcp** the files from the remote host to the SRM, or use NFS to mount the remote host file system onto the SRM.

2. **Cube ownership**
   Cubes that you own are not automatically released when you log out from the remote workstation. You must use **relcube** to release the cube.

3. **Maximum of seven cube partitions per remote host**
   Some remote host operating systems (for example, Sun OS 3.5) support a maximum of only seven cube partitions per remote host. Attempting to allocate an eighth cube may cause the remote host to hang. To recover, reboot the remote host and cube.

4. **setenv TTY 'tty'**
   The TTY environment variable is used by several cube commands and must be set before you use the cube. Because each newly created window on a Sun workstation inherits its parent's shell variables, you should set TTY in *.cshrc* and not in *.login*. However, for the remote copy command (**rcp**) to work properly, you must redirect the standard error from the **setenv TTY 'tty'** to null as follows:

   ```
   setenv TTY 'tty' >& /dev/null
   ```

5. **cubeinfo's SRM field does not contain complete name**
   On a remote host, the **cubeinfo** command's *srmname* field contains the SRM name specified by **getcube -h**, which may not be the complete name.

6. **Heavy message passing**
   Heavy message passing between a remote host and the nodes for a sustained period of greater than 15 hours may cause the remote *commser* process to abort. To recover, execute a **bootcube** on the remote host as root and release the cube on the SRM.

7. **FORCE TYPE range messages**
   Message types in the FORCE TYPE range (types that are greater than 40000000 hexadecimal), sent from a node to a remote host, must be received using a **csendrecv()** or **isendrecv()** call in the host program. Forced messages sent from a node to a **crecv()** or **irecv()** on a remote host will be lost. Force messages can be sent from host program to host program, and received using **crecv()** or **irecv()**. Force messages can also be sent from node program to node program, and received using **crecv()** or **irecv()**.

8. **Merged error messages**
   On a remote host, iPSC/860 error messages may be merged with prints from user applications.

9. **Invalid buffer pointers**
   Invalid buffer pointers in a C host program are sometimes accepted as valid and the system does not return an error message. This may cause unpredictable behavior

DECON CONCURRENT DEBUGGER

1. **DECON may not be able to distinguish between certain variables**
   DECON may not be able to distinguish between a variable or function of a given name and a variable or function of the same name preceded by an underscore (for example, **x** and **_x**). Similarly, it may not be able to distinguish between a name preceded by a single underscore and one preceded by a double underscore.

2. **Register variables in C**
   You may display or assign a register variable only if the execution point is in the scope of the routine where the register variable is defined.

3. **Compiler optimizations may limit access to variables**
   Some compiler optimizations are not turned off by **-B** (this switch is required when compiling a program that is to be debugged by DECON). Consequently:

   - You can't access variables that are never used. The compiler optimizes them away.
   - You can't access subroutine arguments that are passed by reference.

   Compiler optimizations may limit access to variables. DECON may have difficulty accessing Fortran IMPLICIT variables and loop counters on all 386-based nodes (CX, SX, VX). It may appear that you have accessed a loop counter when you actually *have not*. This is not a problem on RX nodes.

4. **Unsupported features**
   This release of DECON does not support the following features:

   - Data breakpoints on host and node processes
   - Label breakpoints on host and node processes
   - Assigning and displaying C bit variables
   - Assigning and displaying Fortran character variables

5. **type command limitations**
   Fortran data types are mapped onto C language data types. For example, the type of a Fortran **character*n** variable is displayed as **character**. The type of a **logical*1** Fortran variable is displayed as **character**, and the type of a **logical*4** is displayed as **integer*4**.

6. **Interrupt while host process is doing a system call**
   If you interrupt a host process while it is executing a **csend()**, **crecv()**, **cprobe()**, or **msgwait()** system call, the host process will hang.

7. **Fortran array starting index**
   DECON assumes that the first element has index 1 for Fortran arrays and 0 for C arrays. If you have a starting index other than 1 in your Fortran program, you need to adjust the index to 1 so that **assign** and **display** can identify the correct element.

8. **Maximum pathname**
   The maximum length of the pathname used in the **source** command is 32 characters.

9. **unset and unalias**
   Both **unset all** *and* **unalias all** delete all aliases *and* all variables.

10. **Accessing large arrays**
    Arrays with more than 64K elements in a dimension cannot be displayed with predictable results.

11. **Cube size limitations**
    DECON is not reliable on cubes larger than eight nodes.

12. **Erroneous error message**
    The following error message may appear inappropriately:

    ```
    error: stopped at different statements
    ```

    Ignore the message, and use the appropriate commands (**break, display, list**) on one process at a time to verify correct operation.

13. **Termination after errors**
    DECON doesn't usually recognize when a node program receives an exception, and does not cause normal error exit processing to occur. Use the **<Del>** key to get back to the DECON prompt, if necessary.

14. **alias limitation**
    Using a debug variable to define an alias aborts DECON.

15. **assign limitations**
    - Avoid prompted assigns for more than one array element per assign command. In some cases, DECON stores an entered value in the wrong array element.
    - Complex variables must be assigned in prompted mode, not on the **assign** command line.
    - In prompted assigning of complex variables, pressing **<Enter>** does not retain the variable's current displayed value, as with other types. Always enter complex variable values when prompted for them, even if you do not want to change the value.
    - When entering character values, enclose them in single quotes.

16. **break limitations**

- In Fortran host programs, set breakpoints on the line following a **GOTO** *destination* label, instead of on the labeled line itself. The actual jump point may compile to a location just after the address of the label.
- You cannot change the context of a breakpoint using the **break** command. You cannot set a breakpoint on an object if a breakpoint for that object already exists in another context. Remove the first breakpoint and define a new one that includes both contexts.
- Do not set breakpoints on Fortran **END** statements.
- In some cases, DECON may not be able to break at the first statement of a program.

17. **display limitations**

- Displaying whole C structures for multiple nodes may show incorrect values. Choose one member at a time (on any number of nodes) or one node at a time (when displaying more than one member).
- Fortran logical variables are not displayed with true/false interpretation, `1` is `.TRUE.` and `0` is `.FALSE.`.

18. **load limitations**
Using multiple **load** commands for multiple processes with the same *pid* on different nodes may cause premature termination of the program or corrupt the processes' contexts. One symptom of this occurrence is that you might receive the following inappropriate error message:

```
error: illegal nodeid nn
```

To restart or reload your application, exit DECON, then invoke it again.

19. **source limitation**
To set the source directory back to the current directory, use the pathname of the current directory (full or relative) or a period enclosed in single quotes. For example:

```
source '.'
```

Without the quotes, DECON may consider the directory file to be the source file for listings.

20. **wait limitation**
When used with **step** commands, the **wait** command may hang. An interrupt should return a DECON prompt.

21. **Only use the stop command on nodes that are running**
Do not use the **stop** command to halt nodes that are not running.

iPSC®/2 SIMULATOR

1. **Process creation**
   The simulator creates one UNIX/XENIX process for every simulated node or host process. Therefore, the size of a simulation is limited by the maximum number of processes allowed by the operating system being used. XENIX software allows 14 processes to be created; and UNIX 4.2 BSD, UNIX 5.2 ATT, and UNIX 5.3 ATT allow 23 processes to be created. Limit the cube's dimension or the number of processes per node to conform to these limitations.

2. **CFS calls and three node calls not supported**
   The CFS calls (such as **cread()**, **cwrite()**, **iowrite()**, **iodone()**, and **iomode()**) are not supported by the simulator. The NX **handler()**, **dclock()**, and **hwclock()** calls are not supported either.

3. **Clock calls and timing**
   Note that all timing uses a common time base. Because the simulated processes are executed in sequential timeslices by UNIX software, the values from the clock are different from those obtained on the iPSC/860 system.

4. **Remote host and cube sharing**
   Remote host and cube sharing functions are not supported by the simulator. Related calls (such as **getcube()**) should be commented out before you compile.

5. **System Resource Manager (SRM) programs**
   System Resource Manager programs are started within the simulator. Thus, the command line cannot be used to supply arguments to the SRM programs or redirect their standard input or output.

6. **File descriptors**
   File descriptors 9, 10, 11, and 12 are reserved for the simulator.

7. **Signals**
   Signal number 28 in the UNIX 4.2 BSD environment; and signal number 16 on XENIX, UNIX 5.2 ATT, and UNIX 5.3 ATT are reserved for the simulator.

8. **Interchanging calls**
   The simulator accepts all host and node calls in both host and node programs, whereas the iPSC system does not. Therefore, it is possible to write simulator programs that will not run on the iPSC system. Refer to the *iPSC®/2 and iPSC®/860 Programmer's Reference Manual* for more information on the calls supported on the host and nodes.

OBJECT FILE UTILITIES

The command **nm860 -x -e** prints addresses in base 10 instead of base 16. The command **nm860 -x** prints addresses in hex as it should.

UNIX SOFTWARE

1. **Cannot use cpio with noclobber set**
   You cannot have **noclobber** set in your environment when you wish to use **cpio** to write to a disk device.

2. **Forcing a script to execute in sh**
   To force a script to execute in **sh**, the first line of the script must be ":".

3. **Forcing a script to execute in csh**
   To force a script to execute in **csh**, the first line of the script must be "#!". (UNIX System V version 3.0 software required only a "#" as the first character in the file.)

4. **unsetenv not supported**
   The **unsetenv** command is not supported in UNIX System V version 3.2. **unsetenv** is not included in the **csh**.

5. **Duplicate symbol defined in include files**
   The symbol `SYMESZ` is defined in both */usr/include/syms.h* and */usr/include/ldfcn.h*. Including both of these files in your application causes a compiler warning.

TCP/IP ON THE SRM

1. **telnet problem**
   telnet will not work from a virtual terminal.

CDP DIAGNOSTIC PROGRAM

The Cube Diagnostic Program (CDP) does not test CMC LAN boards installed in the cube. However, these boards can be exercised manually with the UNIX **ping** command on the SRM. The **ping** command sends an echo request data packet to the specified host on the LAN network. If the network host is functioning, it will respond to the request. The **ping** command will then report statistics indicating the number of packets transmitted and received and the percentage of packet loss. A zero percent packet loss indicates a successful **ping** operation. To exercise the SRM and cube LAN connections, perform the following procedure:

1. Verify the SRM's LAN connection by making the SRM **ping** itself. Enter the following:

   ```
   ping localhost 64 1
   ```

   The **ping** statistics should indicate a 0% packet loss.

2. Examine the */etc/hosts* file on the SRM and locate the entries for the CMC LAN board connections in the cube. These entries have the format *sysname*_**IPSC**_*n*, where *sysname* is the LAN network name of the SRM and *n* is the slot number minus 1 of the CMC LAN board in the system. If this file does not match your system configuration, run the *ipconf* utility. Refer to the *iPSC®/2 and iPSC®/860 System Administrator's Guide* for details.

3. Boot the cube to initialize the CMC LAN boards in the system. Enter

   ```
   bootcube
   ```

4. For each CMC LAN board installed in the system, verify the connection by making the SRM ping the node. Enter

   ```
   ping sysname_IPSC_n 64 1
   ```

   where *sysname* is the LAN network name of the SRM and *n* is the slot number minus 1 of the CMC LAN board in the system. The **ping** statistics should indicate a 0% packet loss.

THE APPLICATION BINARY INTERFACE 3

INTRODUCTION

This chapter contains information pertinent to i860 software developers, especially those writing in assembly language.

Intel Scientific Computers uses the register usage and calling conventions and the data alignment rules of the Application Binary Interface (ABI). The ABI is a hardware and software standard that is intended to allow programs to run without recompiling or relinking on any machine adhering completely to the standard. The ABI standard replaces the previous convention, which is referred to as the PRM (Programmer's Reference Manual) compliant binary convention.

To ensure ABI compatibility among the various i860 environments, a set of binary conventions have been established. This standard defines calling and data alignment conventions.

Existing i860 PRM-compliant NX/2 application binaries need to be converted to the i860 ABI. This chapter describes the differences between the i860 PRM and the i860 ABI and the modifications that are required to convert a PRM compliant application into an ABI compliant application.

NOTE

Chapter 8 of the *i860™ 64-Bit Microprocessor Programmer's Reference Manual* (240239-002), published in 1989, documents the PRM-compliant calling and data alignment conventions. New manual releases, starting with 240239-003, published in 1990, reflect the new ABI calling and data alignment conventions.

CALLING CONVENTIONS

The calling conventions consist of the register allocation and parameter passing conventions. The difference between PRM and ABI are described in Tables 3-1 and 3-2.

Table 3-1. Integer Register Allocation Changes

	PRM Allocation	ABI Allocation
Parameters and Temporaries	r16-r28	r16-r27
Return values	r16	r16-r17
Temporaries	r28-r30	r30-r31, r28 and r29 (if they are not used for argument or environment pointers)
Memory argument area pointer	--	r28
Environment pointer	--	r29

Table 3-2. Floating Point Register Allocation Changes

	PRM Allocation	ABI Allocation
Local values	f2-f15	f2-f7
Parameters and Temporaries	f16-f27	f8-f15
Return value	f16-f17	f8-f11
Temporaries	f28-f31	f16-f31

There are two types of parameter-passing changes.

- The first change occurs when there are more parameters than registers.
- The second change occurs when there are mixed mode parameters.

If there are more parameters than the number of parameter registers, the extra parameters go on the top of the stack in the PRM register allocation scheme. In the ABI allocation scheme, the extra arguments are placed in memory and the integer register r28 (called the memory argument pointer) is set to point to this argument area in memory. This memory may either be statically allocated or allocated off the stack.

The other type of parameter passing change involves the passing of mixed mode parameters. In PRM, the passing of mixed integer and floating point parameters in registers is treated as a special case. If parameter number N is an integer parameter, it goes into the integer register 16 + N, and the floating point register pair at 16 +2N is not used for passing parameters. If parameter number M is a floating point value, it goes into the floating point register pair at 16 + 2M, and the integer register 16 + M is not used for passing parameters. In ABI, no such register skipping is required for mixed integer and floating point parameter passing. The integer and floating point registers are treated independently. For example, if Parm1 and Parm2 are integer variables and Parm2 and Parm4 are floating point variables, the register allocations are as shown in Fig. 3-3.

Table 3-3. Mixed Mode Parameter Allocation

PRM			
Integer Registers		**Floating Point Registers**	
Parm1	r16		f16, f17
	r17	Parm2	f18, f19
Parm3	r18		f20, f21
	r19	Parm4	f22, f23
ABI			
Integer Registers		**Floating Point Registers**	
Parm1	r16	Parm2	f8, f9
Parm3	r17	Parm4	f10, f11
	r18		f12, f13
	r19		f14, f15

DATA ALIGNMENT CONVENTIONS

The alignment requirements of the PRM and ABI conventions differ for structures, unions, and Fortran common blocks. The PRM requires structures, unions, and common blocks to be aligned on16-byte boundaries. But in ABI, the most restrictive alignment from among the members of structures and unions is used as the alignment. The most restrictive alignment for ABI is eight bytes, rather than 16 bytes. However, alignment on 16-byte boundaries is still recommended because it usually results in higher performance.

CONVERSION

The calling and data alignment changes need to be made to the source code of existing assembly language applications. The code must then be reassembled using the Intel assembler.

High-level language applications must be recompiled and relinked with the new compiler and linker.

SOFTWARE INSTALLATION 4

INTRODUCTION

This chapter describes how to install the Release 3.2 software.

CAUTION

If you are an iPSC/2 Ada user, *do not* install this Release 3.2 software on your system until you have the iPSC Ada Release 2.0. The iPSC/2 Release 1.0 Ada libraries are not compatible with Release 3.2 software, but Release 2.0 Ada libraries are.

You need to install the iPSC Release 3.2 system software **only** if the software on your system becomes damaged. If that happens, please call iSC Customer Support before performing any installations:

1-800-421-2823 (Customer Support Hotline)
(44) 793 641 469 (in England)
Your Local Intel Sales Office (in Europe)
support@isc.intel.com (Internet address)

NOTE

Before performing any procedure in this chapter, read through the procedure completely to make certain that you understand what the procedure involves. If you have any questions before or during any procedure, please call iSC Customer Support for help.

If you perform the procedures in this chapter exactly as described, you will create a software system that is identical to the one originally installed by iSC.

The procedures in this chapter use the conventions described in the preface. You should also be aware of the following conventions used in these procedures:

- The instruction "Enter ***character(s)***" means type the indicated character(s), and then press the **<Enter>** key. For example, "Enter *y*" means type the letter y, and then press the key labeled Enter.
- In prompts, square brackets surround a default value. Pressing **<Enter>** selects the indicated default value.
- Some steps in these procedures cause a lot of information to be displayed on the system monitor. However, the step usually shows only the last message displayed. Do not be concerned if the indicated message does not appear immediately. Be patient. Some steps take several minutes to complete.

UPGRADING YOUR SYSTEM SOFTWARE

If you are upgrading your software from a release prior to Release 3.0, you will need to reinstall UNIX and all of the iPSC system software and your iPSC options. This will require that you back up any applications on the disk drive and reformat your SRM hard drive before installing the software as described in this chapter. For more information, contact iSC Customer Support.

If you are upgrading your software from Release 3.0 or later to Release 3.2, it is only necessary to remove the software you are replacing and install the new release as described in this chapter. Intel recommends that you remove packages in the following order:

1. Remote Host Software (optional)
2. iPSC System Software
3. NFS Software (optional)
4. Ethernet drivers
5. TCP/IP Software

The procedure for removing packages is as follows:

1. Login as root at the system console
2. Enter single user mode. This prevents other users from logging in.

```
shutdown -is
```

3. Mount the user partition. The **removepkg** command resides in this partition. The single-user shutdown does not automatically mount the user partition.

```
/etc/mount /dev/dsk/0s3 /usr
```

4. Invoke **removepkg**.

```
/usr/bin/removepkg
```

5. From the menu, choose the package to be removed.

6. After the package is removed, press **<Esc>** to get the root prompt back. D*o not* press **<Enter>**. Pressing **<Enter>** results in a shutdown.

7. Repeat steps 4 through 6 for each package to be removed.

After the desired packages are removed, install the system software as described in this chapter. Installation instructions for options are described in the release notes for that product.

INSTALLING UNIX SYSTEM V/386 RELEASE 3.2 VERSION 2.1

CAUTION

Installing the UNIX software reformats the hard disk. To prevent loss of data, save all user files and any system files that have been modified *before* installing the UNIX software.

If you are upgrading from UNIX Release 3.0, *do not replace the new system files with the modified system files* that you saved. Doing so downgrades those system files to UNIX Release 3.0. Instead, after installing the new UNIX software, make the same modifications to the new system files.

There are six major steps to installing the UNIX software:

1. Install the root and user filesystems
2. Install the base UNIX operating system
3. Install the Cartridge Tape Utilities
4. Install the UNIX add-on packages
5. Set up user and system logins

6. Install the remote terminal package

To install the UNIX software, you must perform all parts in the order specified.

Installing the Root and User Filesystems

Installation Time:	Approximately 30 minutes.
Installation Medium:	Floppy diskette (1) labeled "Boot Floppy."
Information you need:	Size of your hard disk (140MB or 380MB).

1. If the system is on, turn it off as described in Chapter 2 in the sections "Stopping the UNIX Operating System" and "Removing Power."
2. Turn the system on, and quickly insert the installation diskette into the diskette drive.
3. When the following message appears:

   ```
   Strike ENTER to install the UNIX System on your hard disk.
   ```

   Press **<Enter>**.
4. When the following message appears:

   ```
   WARNING: A new installation of the UNIX System will destroy
   all files currently on the system.  Do you wish to continue
   (y or n)?
   ```

   Enter *y*.

5. When the following message appears:

```
SELECT ONE OF THE FOLLOWING:

   1.  Create a partition
   2.  Change Active (Boot from) partition
   3.  Delete a partition
   4.  Exit (Update disk configuration and exit)
   5.  Cancel (Exit without updating disk configuration)
Enter Selection:
```

   Enter *5*. (The partitioning was done when your system was originally installed.)

6. When the following message appears:

```
A surface analysis will now be done.
This will destroy all data on the hard disk.
Strike ENTER to continue or DEL to abort.
```

   Press **<Enter>**.

7. Following a message describing a suggested partitioning, this message appears:

```
Is this allocation acceptable to you (y/n)?
```

   Before responding to this message, make a note of the suggested number of cylinders for the root file system, and for the swap/paging area. You will use these to answer the questions in steps 10 and 11 below.

   Then, to respond to the allocation question, enter *n*. This keeps the partitioning that was originally sent to you.

8. When the following message appears:

```
Do you wish to have separate root and usr filesystems (y/n)?
```

   Enter *y*.

9. When the following message appears:

```
Do you want an additional /usr2 filesystem (y/n)?
```

   Enter *n*.

10. When the following message appears:

    ```
    How many cylinders would you like for swap/paging (1-xxx)?
    ```

    Enter the number of swap/paging cylinders you noted in step 7 above. Typical numbers are ***109*** for a 140M-byte drive, and ***78*** for a 380M byte drive.

11. When the following message appears:

    ```
    How many cylinders would you like for root (1-xxx)?
    ```

    Enter the number of root cylinders you noted in step 7 above. Typical numbers are ***201*** for a 140M-byte drive, and ***143*** for a 380M byte drive.

12. When the following message appears:

    ```
    Is this allocation acceptable to you (y/n)?
    ```

    Enter ***y*** if the cylinder quantities for root and swap/paging are what you entered.

13. When the following message appears:

    ```
    Reboot the system now.
    ```

    The installation of the root and user filesystems is complete.

14. Remove the diskette from the diskette drive.

To complete the installation of the UNIX software, install the base UNIX operating system, as described in the next procedure.

Installing the Base UNIX Operating System

NOTE

You must install the root and user filesystems (as described in the previous procedure) *before* you install the base UNIX operating system.

Installation Time:	Approximately 20 minutes.
Installation Medium:	Cartridge tape labeled "UNIX System V/386, R3.2 V2.1". Diskette labeled "Cartridge Tape Utilities"
Information you need:	Root password. Install password.

---

15. Press **<Ctrl-Alt-Del>** to reboot the system.

16. When the following message appears:

```
Installation from Cartridge Tape is available using Interrupt
#5 and Address Range 300 through 301. Are you installing from
tape (y/n)?
```

Enter *y*.

17. When the following message appears:

```
Please make sure your Cartridge Tape hardware is configured
correctly. Insert System Installation Tape in drive and press
<RETURN>
```

A. Insert the installation tape into the tape drive (label up, exposed tape to the left; push until the tape cartridge locks into place).

B. Press **<Enter>**.

18. When the system prompts for a root password, enter the password that you have chosen for the root login.

19. When the system prompts for an install password, enter the password that you have chosen for the install login.

20. When the following message appears:

```
Strike ENTER when ready
or ESC to stop.
```

    Press **<Esc>**.

21. When the following message appears:

```
Reboot the system now
```

    Press **<Ctrl-Alt-Del>** to reboot the system.

22. When the following message appears:

```
Console login:
```

    Login as root.

23. Enter ***installpkg***.

24. Insert the Cartridge Tape Utilities installation diskette into the diskette drive and turn the handle down.

25. When the following message appears:

```
Strike ENTER when ready
or ESC to stop
```

    Press **<Enter>**.

26. When the following message appears:

```
Type the interrupt number and strike the ENTER key or type Q
to cancel installation.
```

    Enter *5*.

27. When the following message appears:

```
Strike ENTER when ready
or ESC to stop
```

    Press **<Enter>**.

28. Again, when the following message appears:

```
Strike ENTER when ready
or ESC to stop
```

    Press **<Enter>**.

29. When the following message appears:

```
Reboot the system now.
```

    A. Remove the Cartridge Tape Utilities diskette from the diskette drive.

    B. Leave the tape in the tape drive for the next procedure.

    C. Press **<Ctrl-Alt-Del>** to reboot the system.

30. When the following message appears:

```
Console login:
```

    The installation of the base UNIX operating system is complete.

To complete the installation of the UNIX software, install the UNIX add-on packages, as described in the next procedure.

Installing the UNIX Add-On Packages

NOTE

You must install the base UNIX operating system (as described in the previous procedure) *before* you install the UNIX add-on packages.

Also, if any of the UNIX Add-On packages are currently installed, you must remove them before installing the new software. Refer to the section "Remove Add-On Software Packages" in the *AT&T UNIX System V/386 Release 3.2 System Administrator's Guide.*

Installation Time:	Approximately 45 minutes.
Installation Medium:	Cartridge tape labeled "UNIX System V/386, R3.2 V2.1."
Information you need:	Root password.

---

31. Login as root.

32. Enter ***installpkg***.

33. When the following message appears:

```
Are you installing from tape (y/n)?
```

    Enter *y*.

34. When the following message appears:

```
Insert Installation Tape in drive and press <RETURN>.
```

    (If you removed the tape after the last procedure, reinsert the tape into the tape drive.)

    Press **<Enter>**.

35. When the following message appears:

```
Do you want to install all of the above packages? <y/[n]>:
```

Press **<Enter>** (to accept the default *n*).

36. The system then prompts you to select the packages to install. Enter *y* after each of the following packages, and *n* after all the other packages (or press **<Enter>** to accept the default *n*):

```
Editing Package Version 2.0
Extended Terminal Interface Package Version 2.0
C Software Development Set 4.1.6
Network Support Utilities Package (1.2) Version 2.0
2 Kilobyte File System Utility Package Version 2.0
Kernel Debugger(s) - Version 2.0
PC586 Ethernet Driver - Version 1.0
System Administration Software
```

A. When the Kernel Debugger package is being installed, the following appears:

```
Which kernel debugger(s) do you want to install?

   1) DEBUGGER (polish calculator style)
   2) GDEBUGGER (traditional)
   3) both DEBUGGER and GDEBUGGER

Choose 1, 2, or 3
```

Enter *1*.

B. When the System Administration Software package is being installed, the following appears:

```
Do you want to give passwords to administration login?
(y/n) [n]
```

Press **<Enter>** (to accept the default *n*).

37. When the following message appears:

```
Strike ENTER when ready
or ESC to stop.
```

Press **<Enter>**.

38. When the following message appears:

```
Reboot the system now.
```

Press **<Ctrl-Alt-Del>** to reboot the system.

39. When the following message appears:

```
Console login:
```

The installation of the UNIX add-on packages is complete.

40. Remove the tape from the tape drive (push tape in to release catch).

To complete the installation of the UNIX software, set up the user and system logins, as described in the next procedure.

Setting Up User and System Logins

NOTE

You must install the UNIX add-on packages (as described in the previous procedure) *before* you set up the user and system logins.

Installation Time:	Approximately 10 minutes, depending on number of user logins being set up.
Installation Medium:	None.
Information you need:	Name of your time zone. Whether you observe daylight savings time. User login names, IDs, passwords, etc. Administrative passwords. System passwords. *node name* (name by which other machines know this machine).

---

41. Login as setup.

42. Answer the questions that setup asks.

NOTE

One of the questions asks you to select your time zone from a list. If your time zone is not included in this list, select GMT, and then enter the correct local time when a later question asks for the time.

To complete the installation of the UNIX software, install the remote terminal package, as described in the next procedure.

Installing the Remote Terminal Package

NOTE

You must set up the user and system logins (as described in the previous procedure) *before* you install the remote terminal package.

Installation Time:	Approximately 5 minutes.
Installation Medium:	Floppy diskette (1) labeled "Remote Terminal Package."
Information you need:	Root password.

---

43. Login as root.
44. Enter ***installpkg***.
45. When the following message appears:

```
Are you installing from tape (y/n)?
```

    Enter ***n***.

46. Insert the installation diskette into the diskette drive, and turn the handle down.
47. When the following message appears:

```
Strike ENTER when ready
or ESC to stop.
```

    Press **<Enter>**.

48. When the following message appears:

```
Enter option:
```

    Enter ***1***.

49. When the following message appears:

```
Enter a file name, 'all', 'done', or 'files':
```

Enter ***all***.

50. When the following message appears:

```
Enter a file name, 'all', 'done', or 'files':
```

Enter ***done***.

51. When the following message appears:

```
Enter option:
```

Enter ***0***.

52. Remove the diskette from the diskette drive.

53. If you are not planning to install the TCP/IP and Ethernet drivers (as described in the next section), press **<Ctrl-D>** to log out.

This completes the installation of the UNIX software.

INSTALLING TCP/IP AND ETHERNET DRIVERS

NOTE

The UNIX software must be installed *before* you can install the TCP/IP software. Refer to the section entitled "Installing UNIX System V/386 Release 3.2 Version 2.1" in this manual.

Also, if the TCP/IP and Ethernet drivers packages are currently installed, you must remove them before installing the new software. Refer to Section 4.2, "Removing TCP/IP," on page 1-6 of the *Intel® TCP/IP for System V/386 Administrator's Guide and Reference.*

TCP/IP software *must* be installed *before* the iPSC Release 3.2 system software. If you install the TCP/IP software *after* the iPSC Release 3.2 system software package has been installed, you *must* re-install Release 3.2 System software.

Installation Time:	Approximately 30 minutes.
Installation Media:	Floppy diskettes (3) labeled "Intel TCP/IP System V/386 Release 3.2." Floppy diskette (1) labeled "Ethernet Drivers Release 3.2."
Information you need:	Root password. Internet address. Network name. Whether network is subnetted. Domain name.

CAUTION

The *remove* and *install* scripts for TCP/IP remove/overwrite the following configuration files:

/etc/hosts
/etc/networks
/etc/gateways
/etc/hosts.equiv
/usr/lib/named/named.hosts
/usr/lib/named/named.local
/usr/lib/named/named.rev
/usr/lib/named/named.soa
/usr/lib/named/named.cashe

If you are updating a previous release of Intel TCP/IP, you may want to rename or backup these files *before* removing the previous release. After you successfully install TCP/IP Release 3.0, you can restore these files.

Perform the installation procedure described on page 1-6 of the *Intel® TCP/IP for System V/386 Administrator's Guide and Reference* in the section entitled, "Sample Installation: A Three System Local Area TCP Network."

INSTALLING THE RELEASE 3.2 SYSTEM SOFTWARE

NOTE

You *must* install the TCP/IP software (as described in the *Intel TCP/IP for system V/386 User's Guide and Reference*) *before* you install the Release 3.2 system software. Refer to the section entitled "Installing TCP/IP and Ethernet Drivers" in this manual.

Also, if the iPSC/2 software is currently installed, you must remove it before installing the new software. Refer to the section "Remove Add-On Software Packages" in the *AT&T UNIX System V/386 Release 3.2 System Administrator's Guide*.

The install script modifies the three standard *crontab* files and creates a new **/usr/bin/man** command. If you want to keep your existing *crontab* files or **man** command, save them before performing this procedure, and then restore them afterwards.

You *must* install the iPSC/2/860 System S/W R3.2 Update 9/90 software to complete the Release 3.2 system software installation.

Installation Time:	Approximately 25 minutes.
Installation Medium:	1 cartridge tape labeled "iPSC®/2 System Software R3.2"
Information you need:	Root password.

---

1. Login as root.
2. Enter ***installpkg***.
3. When the following message appears:

```
Are you installing from tape (y/n)?
```

   Enter *y*.

4. When the following message appears:

```
Insert Installation Tape in drive and press <RETURN>.
```

   Insert the installation tape labeled "iPSC®/2 System Software R3.2" into the tape drive (label up, exposed tape to the left; push until the tape cartridge locks into place).

5. Press **<Enter>**.

6. Eventually, the following message appears:

```
Do you want to install all of the above packages? <y/[n]>:
```

   Enter *y*.

7. When the following message appears:

```
Strike ENTER when ready or
ESC to stop
```

   Press **<Enter>**.

8. When the following message appears:

```
Reboot the system now.
```

   A. Remove the tape from the tape drive (push the tape in to release the catch).

   B. Press **<Ctrl-Alt-Del>** to reboot the system.

9. When the following message appears:

```
Console login:
```

   The installation of the iPSC/2 Release 3.2 system software is complete.

INSTALLING THE RELEASE 3.2 EXTENSION SOFTWARE

NOTE

You *must* install the iPSC/2 System Software R3.2 software *before* you install the iPSC/860 extension software from the iPSC/2/860 Extension Software R3.2 tape.

If you have an iPSC/2 system and a 140M-byte disk on your SRM, *do not* install the iPSC/860 extension software from the iPSC/2/860 Extension Software R3.2 tape. This software is unnecessary for the iPSC/2 system, and it requires that the SRM have at least a 380M-byte disk.

You *must* install the iPSC/2/860 System S/W R3.2 Update 9/90 software to complete the Release 3.2 extension software installation.

Installation Time:	Approximately 30 minutes.
Installation Medium:	1 cartridge tape labeled "iPSC®/2/860 Extension Software R3.2"
Information you need:	Root password.

---

1. Login as root.
2. Enter ***installpkg***.
3. When the following message appears:

```
Are you installing from tape (y/n)?
```

   Enter *y*.

4. When the following message appears:

```
Insert Installation Tape in drive and press <RETURN>.
```

Insert the installation tape labeled "iPSC®/2/860 Extension Software R3.2" into the tape drive (label up, exposed tape to the left; push until the tape cartridge locks into place).

5. Press **<Enter>**.

6. Eventually, the following message appears:

```
Do you want to install all of the above packages? <y/[n]>:
```

What you install depends on what you use. To save disk space, do not install the Remote Host or Simulator software on your system unless you need it. You can install this software later if desired. However, you *must* install the iPSC i860 extension software before you can use your iPSC/860 system.

If you don't want to install all of the packages, press **<Enter>** (to accept the default ***n***). The system then prompts you to select the packages to install. Enter ***y*** for the packages you want and ***n*** for the ones you don't want.

When the prompt returns, the installation of the iPSC/860 Release 3.2 extension software is complete.

INSTALLING THE RELEASE 3.2 SYSTEM SOFTWARE UPDATE 9/90

NOTE

You *must* install the iPSC/2 System Software R3.2 and, if applicable, iPSC/2/860 Extension Software R3.2 *before* you install the iPSC/2/860 System S/W R3.2 Update 9/90 tape.

During update installation, root *must not* be running *csh*.

The update software only updates the system software if Release 3.2 is installed on your system. Likewise, the update software only updates the i860 extension software if the extension software is installed on your system.

Installation Time:	Approximately 30 minutes.
Installation Medium:	1 cartridge tape labeled "iPSC/2/860 System S/W R3.2 Update 9/90"
Information you need:	Root password.

---

1. Login as root.
2. Execute the following to put the SRM in the maintenance mode:

   ```
   shutdown -g0 -y -is
   ```

   The following message displays:

   ```
   Type Ctrl-d to proceed with normal startup
   (or give root password for system maintenance):
   ```

3. Enter the root password. The following message displays:

   ```
   Entering System Maintenance Mode
   ```

4. Mount the *usr* partition:

   /etc/mount /dev/dsk/0s3 /usr

   The following message displays:

   ```
   mount -f S51K /dev/dsk/0s3 /usr
   ```

5. Change to the */usr/tmp* directory:

   cd /usr/tmp

6. Insert the "iPSC/2/860 System S/W R3.2 Update 9/90" tape in the SRM tape drive.

7. Copy the contents of the tape to the hard disk:

   cpio -icB -I/dev/rmt/c0s0n

8. Run the installation script:

   ./install

9. When the following message appears, installation of the iPSC/2/860 System S/W R3.2 Update 9/90 software is complete:

   ```
   Console login:
   ```

INSTALLING REMOTE HOST SOFTWARE ON A REMOTE HOST

NOTE

The remote host software package must be installed on the SRM *before* you can install the remote host software on a remote host. Refer to the section "Installing the Release 3.2 Extension Software" in this manual.

Use the following procedure to install the remote host software on a remote host:

Installation Time: Approximately 30 minutes.

Installation Medium: None.

Information you need: Root password.

---

1. Login as root on the remote host.
2. Create a directory on the remote host to contain the remote host source code. This directory can be located wherever you choose.
3. Copy the remote host source code from the directory */usr/ipsc/rhost* on the SRM to the directory you created in the previous step.
4. On the remote host, edit the makefile in the top-level remote host source code directory so that the installation directories are correct for your system:

NOTE

All pathnames must be absolute pathnames (i.e., they must start with `/`).

IPSCDIR Contains daemons and named sockets. The default is */usr/ipsc* and the subdirectories *lib* and *log* will be created.

BINDIR Contains cube binaries. On the SRM these files are in */usr/bin*. Any directory can be used for the binaries, as long as the cube users have it in their search path. The default directory on the remote host is *IPSCDIR/bin*.

LIBDIR Contains *libhost.a*. The default location for the library is in */usr/lib*. Again, any directory can be used as long as the user application makefiles reflect the correct directory.

INCDIR Contains the iPSC/2 include files *cube.h* and *fcube.h*. The default is *IPSCDIR/include*.

5. Kill any existing *commser* and *fserver* daemons on the remote host.
6. Enter ***make*** to build binaries on the remote host.
7. Enter ***make install*** to install the binaries in the prescribed directories.
8. Edit the *srms* file (*IPSCDIR/lib/srms*) to include the names of all SRMs that the remote host may access.
9. Edit the user's *.cshrc* file to include the following line:

```
setenv TTY `tty` >& /dev/null
```

This completes the installation of the remote host software on a remote host.