# DCE for the HP 3000

## HP 3000 MPE/iX Computer Systems

**Edition 2**


**Manufacturing Part Number: B3821-90002**
**E1095**

U.S.A. October 1995

# Notice

The information contained in this document is subject to change without notice.

Hewlett-Packard makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability or fitness for a particular purpose. Hewlett-Packard shall not be liable for errors contained herein or for direct, indirect, special, incidental or consequential damages in connection with the furnishing or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

This document contains proprietary information which is protected by copyright. All rights reserved. Reproduction, adaptation, or translation without prior written permission is prohibited, except as allowed under the copyright laws.

### Restricted Rights Legend

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 DFARS 252.227-7013. Rights for non-DOD U.S. Government Departments and Agencies are as set forth in FAR 52.227-19 (c) (1,2).

### Acknowledgments

UNIX is a registered trademark of The Open Group.

Hewlett-Packard Company
3000 Hanover Street
Palo Alto, CA 94304 U.S.A.

© Copyright 1995 by Hewlett-Packard Company.

# Contents

# Figures

# Tables

# Tables

# Preface

This manual describes the DCE for the HP 3000, it is based on OSF DCE version 1.0.2 source code.

This manual is organized into the following chapters:

Chapter 1, “General Information,” provides information on version identification and components and limitations.

Chapter 2, “Installing DCE/3000 Software,” provides hardware and software requirements, media and preinstallation planning.

Chapter 3, “Configuring DCE Cells,” provides general information on using the DCE configurator tool and options.

Chapter 4, “Programming Notes,” the section provides the architecture of threads on MPE/iX as well as building DCE programs.

Chapter 5, “Programming with Kernel Threads,” provides basic thread creation and management routines,

# 1 General Information

This version of DCE/3000 (version A.01.02) is based on OSF DCE version 1.0.2 source code. It provides the following OSF components for the core services:

- **Remote Procedure Calls (RPC)** — support the development of distributed applications by making requests to remotely networked machines as if they were local. RPCs also implement network protocols used by clients and servers to communicate with each other.
- **Kernel Threads** — supports the interfaces defined in Draft 4 of the POSIX 1003.4a specification, with some exceptions as stated in this document.
- **Cell Directory Service (CDS)** — manages a database of information about the resources in a group of machines called a DCE cell. The database consists of the names of resources and associated attributes.
- **Distributed Time Service (DTS)** — provides synchronized time for the computers in a DCE cell.
- **DCE Security** — provides secure communications through the use of services such as authentication, which guarantees the identity of users, and authorization, which keeps track of user privileges.

In the DCE/3000 version A.01.12, the DCE application library is provided as both an archive library (libdce.a) and an executable library (DCEXL.HPDCE.SYS). The concept of an executable library is like the shared library on HP-UX. If you use the archive library, each application binary will contain its own copy of the DCE routines that it calls directly or indirectly. If you use shared library, all DCE applications can share the single copy of the DCEXL on a system.

## Version Identification

Version information for the individual DCE/3000 components can be obtained by running the Version utility against the DCE program. You will find the product version (**B3821AA** A.01.02 for the domestic version, or **B3822AA** A.01.02 for the international version) and the program version control information at the beginning of the Version output. For example, the following is the output from the Version utility for an RPCD program:

```
:version RPCD.HPDCE.SYS
VERSION B.79.01 Copyright (C) Hewlett-Packard 1987.
All Rights Reserved.

RPCD.DCEPROGS.PTHDCE

SOM #1
$Header: nrt0.s,v 1.12 87/06/08 09:36:52 cary Exp $
B3821AA A.01.02 RPCD-001
LIBPTHD1.0.2-001
DCEmpesrc-003
LIBNCK1.0.2-005
LIBIDL1.0.2-001
ERRNO_WRAPPERS-000
DCEsys802addr-003
DCEioctl-003
LIBSEC1.0.2-002
DCEsignals-001
B0508000/SSICSOCN/$Revision: 1.1 $
[IND]@(#)gethost: A0005000

MAX STACK SIZE: 786432
MAX HEAP SIZE: 40960000
CAPABILITIES: BA,IA,PM,MR,DS,PH
UNSAT PROC NAME:
ENTRY NAME:
LIBRARY SEARCH LIST: othdxl.threads.sys envxl.hpdce.sys
```

# DCE/3000 Components and Files

The DCE/3000 components, their corresponding files, the files size (in sectors), and a description of the files are listed in the following tables, Table 1-1 shows the CDS components.

**Table 1-1 CDS Components**

| Filename | Sector Size | Description |
|---|---|---|
| cdsd.pub.sys | 32 | command file |
| /usr/bin/cdsd | 16 | shell script |
| cdsd.hpdce.sys | 4,309 | program |
| cdscp.pub.sys | 16 | command file |
| /usr/bin/cdscp | 16 | shell script |
| cdscp.hpdce.sys | 2,188 | program |
| cdsadv.pub.sys | 32 | command file |
| /usr/bin/cdsadv | 16 | shell script |
| cdsadv.hpdce.sys | 2,333 | program |
| cdsclerk.hpdce.sys | 12,766 | program |

**NOTE** The file sizes list in these tables are for product **B3822AA**. The sizes may be different for product **B3821AA**.

The DTS components are shown in Table 1-2.

**Table 1-2 DTS Components**

| Filename | Sector Size | Description |
|---|---|---|
| dtscp.pub.sys | 16 | command file |
| /usr/bin/dtscp | 16 | shell script |
| dtscp.hpdce.sys | 688 | program |
| dtsd.pub.sys | 32 | command file |
| /usr/bin/dtsd | 16 | shell script |
| dtsd.hpdce.sys | 1,724 | program |
| dtsnullp.pub.sys | 32 | command file |
| /usr/bin/dts_null_provider | 16 | shell script |
| dtsnullp.hpdce.sys | 174 | program |

**Table 1-2 DTS Components**

| Filename | Sector Size | Description |
|---|---|---|
| dtsntpp.pub.sys | 32 | command file |
| /usr/bin/dts_ntp_provider | 16 | shell script |
| dtsntpp.hpdce.sys | 242 | program |

The Security components are shown in Table 1-3.

**Table 1-3 Security Components**

| Filename | Sector Size | Description |
|---|---|---|
| secd.pub.sys | 32 | command file |
| /usr/bin/seed | 16 | shell script |
| secd.hpdce.sys | 15,815 | program |
| secclntd.pub.sys | 32 | command file |
| /usr/bin/sec_clientd | 16 | shell script |
| secclntd.hpdce.sys | 278 | program |
| seccrtdb.pub.sys | 16 | command file |
| /usr/bin/sec_create_db | 16 | shell script |
| seccrtdb.hpdce.sys | 15,951 | program |
| secadmin.pub.sys | 16 | command file |
| /usr/bin/sec_admin | 16 | shell script |
| secadmin.hpdce.sys | 263 | program |
| rgyedit.pub.sys | 16 | command file |
| /usr/bin/rgy_edit | 16 | shell script |
| rgyedit.hpdce.sys | 2,369 | program |
| acledit.pub.sys | 16 | command file |
| /usr/bin/acl_edit | 16 | shell script |
| acledit.hpdce.sys | 329 | program |
| dcelogin.pub.sys | 16 | command file |
| /usr/bin/dce_login | 16 | shell script |
| dcelogin.hpdce.sys | 107 | program |
| kinit.pub.sys | 16 | command file |
| /usr/bin/kinit | 16 | shell script |

**Table 1-3 Security Components**

| Filename | Sector Size | Description |
|---|---|---|
| kinit.hpdce.sys | 1,396 | program |
| klist.pub.sys | 16 | command file |
| /usr/bin/klist | 16 | shell script |
| klist.hpdce.sys | 331 | program |
| destroy.pub.sys | 16 | command file |
| /usr/bin/kdestroy | 16 | shell script |
| kdestroy.hpdce.sys | 279 | program |

The RPC components are shown in Table 1-4.

**Table 1-4 RPC Components**

| Filename | Sector Size | Description |
|---|---|---|
| rpcd.pub.sys | 32 | command file |
| /usr/bin/rpcd | 16 | shell script |
| rpcd.hpdce.sys | 547 | program |
| rpccp.pub.sys | 16 | command file |
| /usr/bin/rpccp | 16 | shell script |
| rpccp.hpdce.sys | 290 | program |
| idl.pub.sys | 16 | command file |
| /usr/bin/idl | 16 | shell script |
| idl.hpdce.sys | 2,166 | program |
| uuidgen.pub.sys | 16 | command file |
| /usr/bin/uuidgen | 16 | shell script |
| uuidgen.hpdce.sys | 125 | program |

The miscellaneous components are shown in Table 1-5.

**Table 1-5 Miscellaneous Components**

| Filename | Description |
|---|---|
| /etc/dce_config/* | shell scripts for dce_config tool |
| /usr/lib/libdce.a | NMRL |
| DCEXL.HPDCE.SYS | DCE shared library |
| /usr/include/dce/*.h | header files |
| /usr/include/dce/*.idl | idl files |
| /opt/dce local/* | directories for DCE use |

## Domestic and International Version

The DCE/3000 Security component of /usr/lib/libdce.a uses the Data Encryption Standard (DES) algorithm as its default encryption algorithm. Because the United States DOD restricts the export of DES software, DCE/3000 supports two binary versions:

- **Domestic (B3821AA)** — this version contains the DES encryption algorithm and makes it available to user applications; it is available to HP customers in the United States only.
- **International (B3822AA)** — this version strips the DES encryption algorithm, making them available to user applications; it is available to all HP customers.

The International version of the software disables the RPC data protection level *privacy*, disallowing users the ability to encrypt their data in RPCs. If an application specifies the *privacy* level of data protection while using the international version of /usr/lib/libdce.a, the application receives an rpc_s_unsupported_protect_level error. This restriction does not apply to the Domestic version.

## Configuration and Diagnostic Tools

The **dce_config** configuration tool, provided by OSF, is available in the MPE/iX environment. Some MPE/iX commands, such as SHOWPROC and ALTPROC, are enhanced to display information for the threaded tasks under DCE. Glance/XL also displays information about individual threads in a task, however the threads are currently identified by their pin number, not their thread ID within the task. Refer to the Threads section for more details.

# Release Limitations

Some of the limitations described here reflect limitations of OSF DCE 1.0.2, other limitations are specific to this release only.

## Limitations of OSF DCE 1.0.2

- There is no support for application localization (only English is supported), or for application internationalization.

**NOTE**

Localization is defined as making software interfaces appear in the native language of the country in which the software is run. For example, all the user interfaces and messages of an application localized for Italy would be in the Italian language.

Internationalization is defined as enabling an application that is distributed across international boundaries to be localized for users in different countries. For example, an application might consist of a server that communicates with clients in Japan and the Netherlands. The internationalized server could return information in such a form that its clients in Japan could display the information in Japanese, and its clients in the Netherlands could display the information in Dutch.

- The **passwd_import** tool, which imports user account information from /etc/passwd files to the Registry database, does not import the passwords themselves. Therefore, after you have used the **passwd_import** tool to create skeletal DCE accounts in the Registry database, you must use the **rgy_edit** tool to add passwords to those accounts.

### Unsupported Configurations

DCE/3000 (version A.01.02) does not support any of the OSF DCE extended services and configurations, this includes:

- DFS
- X.500 Global Directory Services
- Access to the CDS namespace through the X/Open Directory Service (XDS) and X/Open Object Management (XOM) services
- Diskless Operation
- The “+” character. This is not supported in the MPE/iX HFS name syntax; therefore, GMT files (GMT+0 through GMT+13) cannot be created under /opt/dcelocal/etc/zoneinfo. This could cause the **dtscp** program to interpret your local time incorrectly from the show current time command.

### Interoperability of the Domestic and International Versions

The Domestic and International versions are interoperable with one limitation, Domestic-based application servers or clients that specify the privacy RPC data protection level are not interoperable with servers or clients based on the International version.

Neither the Domestic or International versions of DCE are interoperable with any DCE version that have been built with the DES code omitted. Some DCE ports from other vendors were built in this way in order to meet United Sates export requirements. If you are running a DCE port from another vendor, check with that vendor for details.

## Kerberos Authentication Protocol Compatibility

The DCE Security authentication service implements the Kerberos Version 5 Revision 5 protocol specification. Although Kerberos Version 5 includes backward compatibility support for Kerberos Version 4, DCE Security does not implement this support.

## Security for DCE User Accounts

A user's DCE credentials are not automatically removed by exiting a shell or logging out. Unless you plan to leave background processes running that require your DCE credentials, you should manually remove your credentials before logging out by running the **kdestroy** utility. This makes the system more secure by decreasing the opportunity for someone to gain access to your network credentials.

If you do not use **kdestroy**, DCE credentials are retained in the directory /opt/dcelocal/var/security/creds. To avoid unnecessary disk space usage, inactive credential files should be periodically purged from this directory.

# DCE/3000 versus DCE/9000: Differences

It is assumed that readers have the basic understanding of the differences between MPE/iX POSIX environment and a UNIX based system environment. This section is intended to give end users some key concepts on their first exposure to DCE/3000.

## File Naming Convention

The traditional MPE directory structure is made up for a three-level hierarchy and names for those accounts, groups, and files are all upshifted. The addition of POSIX functionality to MPE/iX makes it possible for DCE programs and utilities to have the same name and location as they are used on an HP 9000. While those DCE programs can be executed in the MPE shell environment, a similar MPE name is also available from the MPE CI environment. Table 1-6 lists some examples of the similarities in names.

**Table 1-6 DCE Program Name Comparisons**

| Shell | MPE CI |
|---|---|
| acl_edit | ACLEDIT |
| rgy_edit | RGYEDIT |
| dce_login | DCELOGIN |
| sec_admin | SECADMIN |
| klist | KLIST |
| rpccp | RPCCP |
| cdscp | CDSCP |

## Managing DCE Daemons

The DCE daemons (**rpcd**, **secd**, **cdsd**, and so on) are run as MPE/iX jobs that log on as manager.sys. If passwords are required, you are prompted for them.

It is important to set the job limit high enough so that all the DCE daemons can logon. For better performance, these daemons should run in the CS queue. Be sure to issue the jobpri cs command at startup, to allow these jobs to run in the C queue.

## Security and Remote Login Utilities

You can use standard MPE remote login commands to perform remote DCE cell administration; there are no DCE-integrated MPE/iX login utilities. However, this exposes the cell administrator's password to network attackers whenever you perform a task on a remote system. If a network attacker obtains the password, the security of the cell's DCE services is compromised. The most secure way to perform the cell administration is to log in locally to each system you need to administer.

## Release Documents

The following lists the documents associated (shipped) with this release of DCE/3000:

- *Introduction to OSF DCE* (ISBN 0-13-490624-1) (HP P/N B3190-90005)
- *DCE Application Environment Specification* — RPC (ISBN 0-13-043688-7) (HP P/N B3190-90011)
- *OSF DCE User's Guide and Reference* (ISBN 0-13-643842-3) (HP P/N B3190-90017)
- *Understanding DCE* (ISBN 1-56592-005-8) (HP P/N B3190-90018)
- *Guide to Writing DCE Applications* (ISBN 1-56592-045-7) (HP P/N B3190-90029)
- *OSF DCE Application Development Reference* (ISBN 0-13-186776-8) (HP P/N B3190-90032)
- *OSF DCE Administration Reference* (ISBN 0-13-186750-4) (HP P/N B3190-90033)
- *OSF DCE Administration Guide — Core Components* (ISBN 0-13-186735-0) (HP P/N B3190-90034)
- *OSF DCE Application Development Guide* (ISBN 0-13-186768-7) (HP P/N B3190-90036)
- *NCS 1.5.1 to DCE RPC Transition Guide* (E0293) (HP P/N B3192-90002)
- *DCE for the HP 3000* (E0295) (HP P/N B3821-90001) *This Document*

# 2 Installing DCE/3000 Software

This section defines the hardware/software requirements, media, preinstallation planning, account structure changes, and the basic installation procedure required for this release of DCE/3000.

## Hardware and Software Requirements

Any system that you want to make a member of a DCE cell must meet certain hardware and software requirements. The system requirements are:

| | |
|---|---|
| **System Type** | Any HP 3000 (Series 900) system. |
| **Operating System** | The system that you want to install DCE/3000 on must be running on MPE/iX release C.50.02 (Express 2, MPE/iX 5.0 push (C.50.02)). |
| **Disk Space** | A minimum of 300,000 sectors of disc space are required to install the DCE/3000 software.<br><br>If the system that you are installing on is to be configured as a DCE/3000 client, a minimum of 50 Mb free space is required on the system volume set.<br><br>If the system that you are installing on is to be configured as a DCE/3000 server, a minimum of 100 Mb free space is required on the system volume set. |
| **Memory** | A minimum of 64 Mb of memory is recommended. |
| **Network Dependencies** | You must have the HP NS Transport product installed. |

## Distribution Media

DCE/3000 is not included on the MPE/iX release/update tapes. The DCE/3000 software is shipped on either of two types of media:

- 8mm DAT tape
- 6250 bpi mag tape

DCE/3000 product installation should be completed only after all other HP 3000 products have been installed or updated.

# Preinstallation Planning

Preinstallation planning involves deciding on how many cells to configure, which systems to include in each cell, and where to run the DCE services (for example, Security, CDS, and DTS). This section provides some decision making guidelines for preinstallation planning.

## Determining Cell Boundaries

Before installation, map the boundaries of your cell by listing the systems that will compose your cell. You may find it practical (or necessary) to divide your site into more than one cell.

Consider the following factors when determining the cell boundaries:

- A major criterion for determining cell boundaries is to include principals that share a common purpose and that have similar privileges.
- Multiple cells require more administrative overhead in setting up and maintenance.
- If you are creating more than one cell for your site, you must determine appropriate cell names to support inter-cell communication.

**NOTE** Set the system clock to be less than 5 minutes apart for all systems that you plan to configure in a cell. This must be done even if you do not plan to configure a DTS server.

## Intercell Communications

To implement intercell communications, you must start at least one Global Directory Agent (GDA) daemon per cell. Because this DCE/3000 release does not support GDA, you need to find a node in your DCE cell environment that will support a GDS server. Check the reference pages of intercell Communications for that DCE node.

## Accounting Structure Change

Most files from the DCE/3000 product tape should be restored to the SUPPORT account, depending on the version:

- **B3821AA** for the domestic version (installation job I00B3821A).
- **B3822AA** for the international version (installation job I00B3822A).

When the installation is complete, the DCE files are moved and new groups and directories are created. Table 2-1 lists the DCE file locations:

**Table 2-1 DCE/3000 File Structure**

| Directory | Description |
|---|---|
| HPDCE.SYS | This group contains all the NM programs for DCE/3000. |
| THREADS.SYS | This group contains the threads library OTHDXL. |
| PUB.SYS | This group contains all the MPE scripts to execute DCE commands from the MPE CI. |
| /usr/bin | This directory contains all the POSIX scripts to execute the DCE commands, and the dce libdce.a library. |
| /opt/dcelocal | This directory and its subdirectories are created for DCE configuration and runtime use. |
| /usr/include/dce | This directory contains the .h and .idl files for developing DCE applications. |
| /etc | This directory contains miscellaneous POSIX files. |
| /usr/temp | This directory contains one file that is used by the security daemon. |

**NOTE** The HFS directory /opt/dcelocal/var/security/creds is used to retain all the DCE credentials for all users. Therefore the ACD on this directory is created to allow all CD (Create Directory Entries) and DD (Delete Directory Entries) permissions to other users.

## Checking the System State

When installing DCE/3000, your system is not required to be in the single user state. However, once you have installed the DCE/3000 software and are reinstalling or updating it, all DCE users must be logged off, and all DCE servers and clients must be stopped. If any DCE file is being accessed during a reinstallation or update, those DCE files may not be reinstalled or updated in a consistent state.

Before installing DCE/3000 software, ensure that the system is running MPE/iX version C.50.02 or later (DCE/3000 requires the OS thread changes in this release of MPE/iX).

## Installation

To install DCE/3000, perform the following steps:

1. Log off all users and log on at the console as MANAGER.SYS.
2. Create the following groups if they do not exist. At the system prompt, enter:

   NEWGROUP HPDCE.SYS
   NEWGROUP THREADS.SYS
   NEWGROUP B3821A.SUPPORT for product **B3821AA**
   or
   NEWGROUP B3822A.SUPPORT for product **B3822AA**

3. Restore the files from your DCE product tape.

**NOTE** Do not use the local option in the RESTORE command. If the local option is used, the DCE files will not be restored correctly.

   At the system prompt, enter:

   FILE TAPE;DEV=TAPE
   RESTORE *TAPE;@.@.@;SHOW

4. Start the product installation job by entering the following:

   STREAM I00B3821.B3821A.SUPPORT for product **B3821AA**
   or
   STREAM I00B3822.B3822A.SUPPORT for product **B3822AA**

5. If the message “DCE has been successfully installed” is displayed at the console, the DCE installation has completed successfully.

   If the message is not displayed or if a warning message(s) is display at the console, the DCE software has probably not completed its installation. Check the spoolfile to find out why it will not complete.

**NOTE** If you currently have DCE configured but stopped, then the /etc/rc.dce file contains the current configuration. The installation job (IOUB3821 or IOUB3822) will not write over this file, instead it renames the new /etc/rc.dce file to /etc/rc.dce.102. It is strongly recommended that you compare your /etc/rc.dce file with the /etc/rc.dce102 file and make any necessary changes.

6. Ensure that the network services are brought up with the appropriate interface name. You must install the DCE software on all of the HP 3000 systems that you plan to configure in your cell. After all of the systems have DCE/3000 installed, you can configure your cell.

   Refer to Chapter 3, “Configuring DCE Cells,” for cell configuration information.

# 3 Configuring DCE Cells

This section provides general information on using the DCE configurator to add your MPE/iX HP 3000 system into a cell. It is divided into two subsections:

- Using the DCE Configuration Tool — provides detailed steps to bring up the DCE Configuration main menu (these steps must be completed each time you change the DCE cell configuration).
- Using the DCE Configuration Options — provides detailed steps for each option in the DCE configuration main menu (basic familiarity with DCE terms and concepts are assumed, as described in the *Introduction to OSF DCE* (B3190-90005).

**NOTE** To configure a cell, you must have previously completed the installation procedure (refer to Chapter 2, “Installing DCE/3000 Software,” for planning and installation information).

# Using the DCE Configuration Tool

The DCE configurator (called **dce_config**) is a shell script based configuration tool, this enables you to run **dce_config** from within the MPE/iX POSIX shell.

Check the following preliminary tasks before you enable the DCE configuration main menu:

- Ensure that the system network is running (RPC requires network sockets).
- Create an MPE/iX group named DCECONFG. At the system prompt, enter:

  NEWGROUP DCECONFIG

  You must be in an MPE/iX group (that is, your working directory must be an MPE/iX group not a POSIX directory) when you start the POSIX shell that runs **dce_config**.

Perform the following steps to obtain the DCE Main Menu for configuring cells:

1. Log on to the console as MANAGER.SYS,DCECONFG. At the system prompt, enter:

   HELLO MANAGER.SYS,DCECONFG

2. Enter the POSIX shell. At the system prompt, enter:

   sh.hpbin.sys -L

   The shell prompt is displayed (for example, shell/iX>).

3. Ensure that /usr/bin is in your shell command search path. At the shell prompt, enter:

   export PATH=/usr/bin:$PATH

4. Bring up the DCE configuration main menu. At the shell prompt, enter:

   dce_config

   The DCE Main Menu as shown in Figure 3-1 is displayed on the console.

**Figure 3-1 DCE Main Menu**

```
DCE Main Menu

   1. CONFIGURE        configure and start DCE daemons
   2. START            re-start DCE daemons
   3. STOP             top DCE daemons
   4. UNCONFIGURE      remove a host from CDS and SEC databases
   5. REMOVE            stop DCE daemons and remove data files created by DCE daemons

  99. EXIT

  selection:
```

From this menu you can configure your system as a DCE server or client system.

# Using the DCE Configuration Options

The DCE configure options allow you to perform multiple tasks on a given DCE cell. This subsection includes the required steps (in order):

1. Configuring an Initial Cell
2. Configuring a DTS Server
3. Configuring a DCE Client (Client-Only System)
4. Removing or Reconfiguring a Client
5. Removing or Reconfiguring a Server

For more information about your configuration options (why and/or when to use them), refer to the *OSF DCE Administration Guide — Core Components* (B3190-90034) document.

## Configuring an Initial Cell

When creating a DCE cell, servers must be configured before clients. Configuration must be performed in the following order:

1. Security server
2. CDS server
3. Time server(s)
4. Time provider

When these server systems have been configured, the client systems can be configured.

To configure an MPE/iX system as the primary server for the core DCE services, perform the following steps:

1. Select “1. CONFIGURE” from the DCE Main Menu, the DCE Configuration Menu as shown in Figure 3-2 is displayed:

**Figure 3-2 DCE Configuration Menu**

```
DCE Configuration Menu

   1. Initial Cell Configuration
   2. Additional Server Configuration
   3. DCE Client

  98. Return to previous menu
  99. EXIT

  selection:
```

2. Select “1. Initial Cell Configuration” from the DCE Configuration Menu, the Initial Cell Configuration menu as shown in Figure 3-3 is displayed:

**Figure 3-3 Initial Cell Configuration**

```
Initial Cell Configuration


   1. Security Server
   2. Initial CDS Server


  98. Return to previous menu
  99. EXIT


  selection:
```

3. Select “1. Security Server” from the Initial Cell Configuration menu.

   If you are re-configuring a cell, answer “Y” to the following question shown in Figure 3-4 (this is always a safe answer).

**Figure 3-4 Reconfiguring Question**

```
... remove all remnants of previous DCE configurations? Y
```

   If this is your first cell configuration, or if you have previously run REMOVE, answer “n” to the question displayed.

4. Enter a cell name, keyseed, cell administrator's principal name, and the principal's password as shown in Figure 3-5.

**Figure 3-5 Entering Information**

```
... enter the name of your cell: my_cell_name


... enter keyseed for initial database master key: <anykey>


...Cell Administrator’s principal name: cell_admin
...password for the Cell Administrator: password
Re-enter desired password: password
```

   Progress messages are displayed from **dce_config** and other programs it invokes. Common messages displayed include “password must be changed” (from the **dce_login**) and “bye” (from **rgy_edit**), these are not errors or warnings.

Security configuration takes approximately five to ten minutes. When complete, three DCE daemon jobs (**rpcd**, **secd** and **secclntd**) are running.

After the Security server has completed configuration, **dce_config** returns to the DCE Configuration menu.

5. From the DCE Configuration Menu, select "1. Initial Cell Configuration". Then select "2. Initial CDS Server" to configure the CDS server.

   This machine creates a cell directory, the namespace is initialized, and ACLs are set for all new namespace entries.

6. Respond to the "..multiple LANs.." question in Figure 3-6:

   - If the DCE cell machines will be on different LANs, respond Y (yes).
   - If the DCE cell machines will be on the same LANs, respond N (no).

**Figure 3-6 Multiple LANs**

```
... Are you using multiple LAN's within this cell? N
```

**NOTE** Failure to answer the "..multiple LANs.." question correctly results in an incorrect network profile and a non functional DCE cell. A "Y" answer is the safest if you are unsure.

If your cell does span multiple LAN's, **dce_config** asks for the name of the LAN as shown in Figure 3-7, where the machine being configured resides. The name you provide is arbitrary, and is used by **dce_config** to store cell profile information.

**Figure 3-7 Name of LAN**

```
... What is the name of the LAN? lan_50
```

CDS configuration takes longer than Security configuration (approximately one hour on small systems). When complete, another two DCE daemon jobs (**cdsadv, cdsd**) are up and running.

## Configuring a DTS Server

DTS servers may be configured on any system in the cell. A minimum of three Time servers is recommended for any cell with three or more member systems. Refer to the *OSF DCE Administration Guide — Core Components* (B3190-90034) for a discussion of the optimum placement of servers in a cell with gateway or WAN links.

If you do not want to configure DTS or if you do not have three systems in a cell, you can skip this section.

**NOTE** Before configuring a DTS server, you must complete the "Initial Cell Configuration" on that system. It is recommended that the system that you plan to add a DTS server to is configured as a DCE client before starting the DTS server configuration.

To configure a DTS server, perform the following steps:

1. Select “2. Additional Server Configuration” from the DCE Configuration Menu. The Additional Server Configuration menu as shown in Figure 3-8 is displayed.

**Figure 3-8 Additional Server Configuration**

```
Additional Server Configuration

   1. Additional CDS Server(s)
   2. DTS
   3. Replica Security Server

  98. Return to previous menu
  99. EXIT

  selection:
```

2. Select “2. DTS” to configure the DTS server. The DTS Configuration Menu as shown in Figure 3-1 is displayed.

**Figure 3-9 DTS Configuration Menu**

```
DTS Configuration Menu
   1. DTS Local Server
   2. DTS Global Server (needed only in multi-LAN cells)
   3. DTS Clerk (needed only when changing back to a clerk)
   4. DTS Time Provider

  98. Return to previous menu
  99. EXIT

  selection:
```

3. Start the DTS daemon:

   - For servers on the same LAN, select “1. DTS Local Server”.
   - For servers that intend to communicate across LAN boundaries, select “2. DTS Global Server”.

     For a discussion about the use of DTS global servers for time servers communicating between LANs, refer to the *OSF DCE Administration Guide — Core Components* (B3190-90034).

     Either selection starts the DTS daemon.

**NOTE** DTS requires at least three servers in order to function. Skipping DTS will not have a direct impact on Security and CDS. However, Security requires that clock skew among systems be no more than five minutes. If the difference is more than 5 minutes, you can use the MPE/iX SETCLOCK command to reset your system clocks on the DTS server systems. Ensure that the system time and TIMEZONE are both set correctly with SETCLOCK.

4. When the Time servers have completed their configuration in a cell, select "4. DTS Time Provider" from the DTS Configuration Menu to configure a DTS time provider on one of the time servers in a cell. The DTS Time Provider Menu as shown in Figure 3-10 is displayed.

**Figure 3-10 DTS Time Provider Menu**

```
DTS Time Provider Menu

   1. Configure a NULL time provider
   2. Configure a NTP time provider

  98. Return to previous menu
  99. EXIT

  selection:
```

**NOTE** A time provider should be configured on one node only within the cell.

The DTS NULL time provider configures a system to trust its own clock as an accurate source of time. The DTS NTP time provider obtains an accurate source of time from other systems outside the cell. Refer to the *OSF DCE Administration Guide — Core Components* (B3192-90034) for more information about time provider.

If you selected "2. Configure an NTP time provider", respond to the question in Figure 3-11.

**Figure 3-11 Hostname Question**

```
Enter the hostname where the NTP server is running: MyHost
```

5. To ensure that all DTS servers are configured correctly, use the following commands:

   shell/iX> dtscp show all
   shell/iX> dtscp show state
   shell/iX> dtscp show local servers

   The show local servers command displays all DTS servers in the cell except for your own system.

6. To display the current time from the dtscp program, setup the following softlink in the shell:

   shell/iX> id /etc/zoneinfo
   shell/iX> in -s US/Pacific localtime *California local time*

   When the localtime softlink has been set, then the time can be displayed with the following command:

   shell/iX> dtscp show current time

### Configuring a DCE Client (Client-Only System)

A DCE client can not be configured without a functional DCE cell. In other words, when you configure your machine as a DCE client, the DCE cell that you are going to configure needs to be up and running. You need to know the name of the cell and the names of the systems that the DCE servers (Security, CDS and DTS) reside.

Before preceding with the DCE Client configuration, ensure that the HOSTS.NET.SYS file in your machine contains the IP addresses for the systems that are running as Servers. When complete, follow the description in the "Startup the DCE Configuration" menu to bring up the DCE main menu.

The following steps enable you to add your machine as a DCE client node:

1. Select "1. Configure" from the DCE Main Menu.
2. Select "3. DCE Client" from the DCE Configuration menu.
3. Respond to the questions as shown in Figure 3-12.

**Figure 3-12 Security Client**

```
   Enter the name of your cell (without /,,,/): n22cell
     .
     .
   What is the name of the Security Server for this cell? server1
     .
     .
     .
   You can either continue or exit from dce_config.
   Do you wish to continue (y/n)? (y): y
   Enter Cell Administrator's principal name: cell_admin
Enter password:    password
     .
     .
     .
   This machine is ow a security client.
```

   Two DCE daemon jobs (**rpcd**, **secclntd**) are streamed and are running. You are informed that your machine is now a Security client.

4. Respond to the questions shown in Figure 3-13 to add CDS client configuration to your system:

**Figure 3-13 Add CDS Client**

```
    .
    .
    .
Continue or exit from dce_config. Do you wish to continue? Y

    .
    .
    .
Enter name of primary CDS server: server1
Can my+machine broadcast to server1?
```

Answer “Yes” if **my_machine** (the name of your machine) is on the same LAN as the remainder of the cell. If you are not sure if they are on the same LAN, respond “No.” An incorrect “No” answer causes a local CDS cache to be set up for the client machine; an incorrect “Yes” answer results in an incorrect network profile and a non-functional DCE cell.

5. Respond to the “...multiple LAN's...” question as shown in Figure 3-14.

**Figure 3-14 Using Multiple LANs Question**

```
Are you using multiple LAN’s within this cell? (n): n
```

One DCE daemon job (**cdsadv**) is now running and you are informed that this machine is now a CDS client.

6. If you want to continue adding your machine as a DTS client, refer to Figure 3-15 and respond with a “Yes” to the following prompt; however, if you are not using DTS within the cell or you want this node to be a DTS server, respond “No” to the prompt:

**Figure 3-15 Continue or Exit**

```
    .
    .
    .
...continue or exit from dce_config. Do you wish to continue? y
```

7. Respond “No” to the ...make this a DFS client prompt. DCE/3000 does not support DFS.

## Removing or Reconfiguring a Client

The procedure described below is used for:

- removing a client
- reconfiguring a client

- stopping a cell
- changing the name of a cell
- changing or modifying a configuration

To remove or reconfigure a client (the client cannot be a Security server or a CDS server), perform the following steps:

1. Bring up the DCE main menu (as described in "Using the DCE Configuration Tool" earlier in this section).
2. Select the "4. UNCONFIGURE" option (this option can be executed from any system in the cell). The UNCONFIGURE option removes the target machine from the cell Security database and the CDS namespace; therefore, do not use the UNCONFIGURE option on a system that is used as a Security server or a CDS server.

**NOTE** DCE client daemons must be running on the system executing the UNCONFIGURE option. If DCE daemons have been stopped, use the START option from the DCE Main Menu to restart the daemons before using the UNCONFIGURE option.

3. The system prompts for the name of the client system to be unconfigured as shown in Figure 3-16.

**Figure 3-16 Unconfigured Node**

```
Enter hostname of node to be unconfigured: my_client
```

**NOTE** If there were any errors unconfiguring the client system, then the client must be unconfigured from another system in the cell.

4. The system prompts for a continuance as shown in Figure 3-17 (unconfiguring a node removes its ability to operate in a cell), you must respond.

**Figure 3-17 Continuance**

```
Do you wish to continue (y/n)? Y
```

The **dce_config** tool deletes the registry entries and CDS entries for the client, then the following message as shown in Figure 3-18 is displayed:

**Figure 3-18 Remove Message**

```
A dce_config REMOVE will need to be performed from node
before reconfiguring it.
```

5. The DCE Main Menu is displayed, select the "5. REMOVE" option on the client system.

   The "5. REMOVE" option stops all running DCE daemons and removes all previous configuration files on the local machine.

## Removing or Reconfiguring a Server

The procedure described below is used for:

- removing a DCE server
- reconfiguring a DCE server
- changing the name of a cell
- changing or modifying a configuration
- restoring a server after a system crash

**NOTE**

If you want to unconfigure the server, do not perform an "UNCONFIGURE", instead perform a "REMOVE" option.

Removing a Security or CDS server requires that you reconfigure the entire cell.

If you are removing both the clients and servers, all client systems must be unconfigured and removed before the server systems are removed. If you want to remove and reconfigure a client, you can do so without reconfiguring the other members of a cell.

To remove or reconfigure a server, perform the following steps:

1. Ensure you are not DCE logged in as a DCE cell principal.
2. Bring up the DCE Main Menu (as described in "Using the DCE Configuration Tool" earlier in this section).
3. Select "5. REMOVE" from the DCE Main Menu. The **dce_config** tool displays the following message as shown in Figure 3-19:

**Figure 3-19 Remove Message**

```
REMOVE will remove the nodes's ability to operate in the cell.
A reconfiguration of the node will be required. if this is not
a server node, then this node should be unconfigured before a
REMOVE is done. Do you wish to continue (y/n)?
```

A "Yes" response stops all running DCE daemons in that system and removes all files created during the initial cell configuration.

# 4 Programming Notes

This section assumes that DCE application developers have some experienced in porting standard C applications to the MPE/iX POSIX environment. For application developers who are not familiar with the MPE/iX POSIX and C language interface, please read the *MPE/iX Developer's Kit* (36430A) first.

# Threads Architecture

This section describes the architecture of threads on MPE/iX.

The following terminology is adopted throughout the remainder of this document. The term *process* refers to the MPE/iX operating system notion of process. The term *task* is defined as a multi-threaded application (depending on the implementation, a task can consist of a single process or multiple processes).

## Threads on MPE/iX

A multi-threaded task on MPE/iX is implemented with multiple processes (one per thread). A task's threads are a cooperative processes in that they share some resources that are normally private to a process. All threads within a task share the same SR 5 space as the initial thread (a process created using run or createprocess). The heap and global variables are shared by all threads, along with loader information and system information regarding open files and sockets.

All other process resources are private to the thread. Each thread has its own NM stack, CM stack, pin number, PIB, PIBX, TCB, PCB, PCBX, process port, and so on. Fields within these data structures that are shared among threads (such as, file system information) are kept in a common location.

## Process Management and Threads

An initial thread is a process created using **run** or **createprocess** (or **fork** and **exec** for POSIX). The threads of a task cannot exist independently of the initial thread. If the initial thread terminates or is killed, all of the task's threads are terminated. A secondary thread cannot be adopted by another task.

Each thread begins execution at an entry point specified at creation time. The entry point is an MPE/iX procedure with one parameter. This procedure resides in either the program file or the linked libraries of the task.

When a thread is created, the following attributes can be specified:

| | |
|---|---|
| **Stack size:** | NM stack size for the thread |
| **Inherit scheduling:** | inherit the scheduling policies of the creating thread |
| **Priority:** | priority of the thread |
| **Scheduling policy:** | round robin, FIFO,... |
| **Scheduling scope:** | priority is global/local |

These attributes are required in order to be POSIX compliant. POSIX also permits each implementation to add its own thread creation attributes. The following attribute was added for MPE/iX:

| | |
|---|---|
| **Debug:** | enter debug before starting the thread |

**NOTE** PH capability is required to create a thread.

From a process management point of view, thread creation is just an abbreviated form of process creation.

All threads are created as siblings. The threads of a task all have the same father task; namely, the father of the initial thread. If a thread creates a child using creatprocess, that child is the child of the task, not of the thread. From the tasks child-point-of-view, its father is the initial thread. When a thread exits, the children and the threads it created are not terminated.

Threads do not “own” the child processes they create. However, threads may find it necessary to wait for the termination of the offspring that they created. Therefore, a thread is permitted to wait for a specific child to terminate and is permitted to wait on the termination of any child. Refer to the **suspend** and **activate** intrinsics for more explanation.

While threads are implemented with multiple processes, to the end user threads should appear to coexist within a single process. Process management hides the MPE/iX implementation of threads from the programmer. The process handling intrinsics work on a task basis.

## Development, Debugging, and Application Execution of Threads

This section discusses the development, debugging, and execution of applications that use threads on MPE/iX. It should be read before attempting to create or run an application that uses threads.

Debug has the following features to facilitate debugging in a threaded environment:

- Breakpoints
- Commands
- Environmental Variable

### Breakpoints.

There are three types of breakpoints available when debugging a threaded program:

| Breakpoint Type | Description |
|---|---|
| **Task-Wide** | Breakpoints that are recognized by any thread within a task. |
| **Thread-Specific** | Breakpoints that are identical to pin-specific breakpoints, but are thread-private, and are specified using an enhanced syntax. |
| **Stop-All-Threads** | Breakpoints with this option, when encountered by a thread within a threaded task suspend all other threads within the task until a CONTINUE command is issued. |

The syntax for the address and pin parameters to breakpoint commands includes the specification:

logaddr [:pin|:@]

and the following for threads:

logaddr [:[[init_thread_pin].tin |.@][:@]]

where *tin* is the thread number returned by **pthread_create**. The pin number of the initial thread can be obtained using SHOWPROC. The syntax [init_thread_pin].*tin* specifies a thread, [init_thread_pin].@ specifies a task-wide breakpoint, and :@ following a

[init_thread_pin].*tin* specification specifies a stop-all-threads breakpoint option.

For example:

| Example Breakpoint | Description |
|---|---|
| B thd_mtx:2e.2 | Sets a breakpoint at **thd_mtx** to be recognized by tin 2 of the task with initial thread 2e. |
| B thd_mtx:.2 | Sets a breakpoint at **thd_mtx** to be recognized by tin 2 of the current task. |
| B start_thread:2c.@ | Sets a task-wide breakpoint at **start_thread** to be recognized by all threads within the task with initial thread 2c. |
| B start_thread:.@ | Sets a task-wide breakpoint at **start_thread** to be recognized by all threads within the current task. |
| B HPFOPEN::@ | Sets a breakpoint at **HPFOPEN** for the current pin (tin) with the stop-all-threads option that is honored if the pin belongs to a threaded task. |
| B HPFOPEN:.3:@ | Sets a breakpoint at **HPFOPEN** for tin 3 of the current task, and the breakpoint has the stop-all-threads option. |
| B HPFOPEN:.@:@ | Sets a task-wide breakpoint at **HPFOPEN** for the current task, and the breakpoint has the stop-all-threads option. |

### Commands

The following commands aid in debugging threaded applications.

| Command | Description |
|---|---|
| **TIN** [init_thread_pin.]tin | This command causes debug to switch to the environment of the specified tin. The default init_thread_pin is that of the current task. Privilege mode is required to switch to any tin in another task. |
| **SUSPEND** | This command suspends all other threads within the task of the tin being debugged. The suspended threads are not resumed automatically with the continue command. |
| **ACTIVATE** | This command resumes the threads that were suspended by the SUSPEND command. It should be issued from the same tin that issued the SUSPEND command. |

### Environmental Variables

There are two environment variables that simplify debugging applications:

| Environment Variable | Description |
|---|---|
| SS_TERM_KEEPLOCK | When set to TRUE, a pin (tin) being debugged retains the terminal semaphore while single-stepping. This prevents any other pin (tin), |

| | |
|---|---|
| | that is waiting to enter debug, from obtaining the terminal semaphore and interfering with the debug session. |
| TERM_KEEPLOCK | Allows a process to retain the terminal semaphore under all conditions until the process terminates or the variable is reset to FALSE. However, this variable has the potential to create a deadlock. For example, a deadlock occurs if the process owning the terminal semaphore waits for another process that in turn is waiting for the debug terminal semaphore. |

### Limitations

The following are know limitations for the debug thread commands:

- The **break** command followed by an abort command hangs the task if the initial thread is waiting to enter debug (such as, another thread is currently in debug).
- The **SUSPEND** command has the potential to hang a task if the user does not issue an **ACTIVATE** command before doing the **CONTINUE** command.
- Each thread has its own debug environment. For example, loaded macros and environmental variables are not shared by threads within a task, and must be dealt with on an individual basis for each thread.

# Building DCE Programs

## Header Files

In addition to the standard POSIX libraries and HP C/XL functions, you may have to include the DCE header files, which can be found in the /usr/include/dce directory. If your C applications use **Try/Catch** for exception handling, you should include the following statement in the C programs:

```
#include <dce/pthread_exc.h>
```

There are no MPE/iX equivalent libraries for /usr/lib/libbb.a or /usr/lib/libc_r.a. The reentrant functions that are defined in MPE/iX and the thread-safe wrapper functions are in /usr/lib/libdce.a.

MPE/iX does not have the file strings.h. The HP-UX strings.h includes string.h, sys/stdsyms.h and some definitions that are strictly for C++ and HP-UX.

## Compiler Flags

When compiling DCE applications using ANSI C under the MPE CI, set the following compiler switches:

```
-D_POSIX_SOURCE -D_MPEXL_SOURCE -D_SOCKET_SOURCE -D_REENTRANT -Aa
```

When compiling under the MPE POSIX shell, you need the above flags except for the -Aa option. If -Aa is set, /bin/c89 displays a large amount of error messages (by definition, the POSIX environment always uses the ANSI C compiler).

## Unresolved Externals

When porting applications from a UNIX environment to MPE/iX, you may receive unresolved external errors during a compile, link, or run phase. It is likely that the unresolved externals are not part of the POSIX.1 standard. To find out if a function is defined in the POSIX environment, look at the manpage for that function on a UNIX system. At the bottom of the manpage, there is a section titled **STANDARD CONFORMANCE**, which lists the function name and the standard it conforms to. If the manpage does not have POSIX.1 listed as one of the standards then that function is not part of the MPE/iX POSIX Environment. To get around this porting issue, you may have to write a routine to emulate the functionality for the unresolved external.

### MPE/iX Makefile Example

The following is an MPE/iX makefile example.

```
#
# (c) Copyright 1992, 1993 Hewlett-Packard Co.
#
# @(#)HP DCE/3000 1.0.2
# @(#)Module: Makefile $Revision:1.1.7.2 $
# $Date:1993/07/08 00:06:21$
# Makefile modified for use on an HP 3000.
#
DEBUG           =
INCENV          = -I. -I/usr/include
ANSI_FLAGS      = -D_POSIX_SOURCE
HP_FLAGS        = -D_REENTRANT -D_MPEXL_SOURCE -D_SOCKET_SOURCE
CFLAGS          = ${ANSI_FLAGS} ${DEBUG} ${HP_FLAGS} ${INCENV}
LDFLAGS         =
LIBS            = -ldce -lsocket -lsvipc -lm -lc
PROGRAMS        = sleeper_server sleeper_client
server_OFILES   = sleeper_sstub.o manager.o server.o
client_OFILES   = sleeper_cstub.o client.o
IDLFLAGS        = -keep c_source ${INCENV}
IDLFILES        = sleeper.idl
IDLGEN          = sleeper.h sleeper_*stub.c sleeper_*aux.c
IDL             = /SYS/HPBIN/SH idl
all:            objects ${PROGRAMS}
objects:        ${server_OFILES} ${client_OFILES}
fresh:          clean all
clean:;
  rm -f ${server_OFILES} ${client_OFILES} ${PROGRAMS} ${IDLGEN}
clobber:        clean
  rm -f a.out core ERRS make.out *~
sleeper_server: ${server_OFILES}
  $(CC) ${LDFLAGS} ${server_OFILES} ${LIBS} -o $@
  mv -f sleeper_server /SYS/PUB/SLEEPSRV
  callci linkedit \"altprog sleepsrv.pub.sys\;xl='othdxl.threads.sys'\"
  mv -f /SYS/PUB/SLEEPSRV sleeper_srver
sleeper_client: ${client_OFILES}
```

```
	$(CC) ${LDFLAGS} ${client_OFILES} ${LIBS} -o $@
	mv -f sleeper_client /SYS/PUB/SLEEPCLT
	callci linkedit \"altprog sleepclt.pub.sys\;xl='othdxl.threads.sys'\"
	mv -f /SYS/PUB/SLEEPCLT sleeper_client
sleeper_cstub.c sleeper_sstub.c sleeper.h:	${IDLFILES}
	$(IDL) ${IDLFLAGS} ${IDLFILES}
sleeper_cstub.o sleeper_sstub.o manager.o server.o client.o: sleeper.h
manager.o server.o client.o: common.h
```

## HP-UX Makefile Example

The following is an HP-UX makefile example.

```
#
# (c) Copyright 1992, 1993 Hewlett-Packard Co.
#
# @(#)HP DCE/9000 1.0.2
# @(#)Module: Makefile $Revision: 1.1.7.2 $
# $Date: 1993/07/08 00:06:21$
# Makefile for use with an HP 9000.
#
DEBUG           = -g
INCENV          = -I. -I/usr/include/reentrant
ANSI_FLAGS      = -Aa -D_POSIX_SOURCE
HP_FLAGS        = -D_REENTRANT -DTRACING
CFLAGS          = ${ANSI_FLAGS} ${DEBUG} ${HP_FLAGS} ${INCENV}
LDFLAGS         = ${DEBUG} -Wl, -a, archive
LIBS            = -lbb -ldce -lm -lc_r
PROGRAMS        = sleeper_server sleeper_client
server_OFILES   = sleeper_sstub.o manager.o server.o
client_OFILES   = sleeper_cstub.o client.o
IDLFLAGS        = -keep c_source ${INCENV}
IDLFILES        = sleeper.idl
IDLGEN          = sleeper.h sleeper_*stub.c sleeper_*aux.c
IDL             = idl
all:            objects ${PROGRAMS}
objects:        ${server_OFILES} ${client_OFILES}
fresh:          clean all
clean:;
  rm -f ${server_OFILES} ${client_OFILES} ${PROGRAMS} ${IDLGEN}
clobber:        clean
  rm -f a.out core ERRS make.out *~
sleeper_server: ${server_OFILES}
  $(CC) ${LDFLAGS} ${server_OFILES} ${LIBS} -o $@
sleeper_client: ${client_OFILES}
  $(CC) ${LDFLAGS} ${client_OFILES} ${LIBS} -o $@
sleeper_cstub.c sleeper_sstub.c sleeper.h:      ${IDLFILES}
  $(IDL) ${IDLFLAGS} ${IDLFILES}
sleeper_cstub.o sleeper_sstub.o manager.o server.o client.o: sleeper.h
manager.o server.o client.o: common.h
```

# 5 Programming with Kernel Threads

Programming with threads, is useful for structuring programs, enhancing performance through concurrency and overlapping I/O, making client/server interaction more efficient, and increases programming complexity. Some things you need to address when programming with threads are:

- Creation and management of threads.
- Threads synchronization and communication.
- Threads scheduling.
- Error handling

A traditional non-threaded process has a single thread of control, started and terminated with the process, and multi-threaded programs require that threads be created and terminated explicitly.

The HP 3000 Kernel Threads Service provides basic thread creation and management routines. Refer to the *OSF DCE Application Development Guide* (B3190-90036) and the *OSF DCE Application Development Reference* (B3190-90032) for detailed information on thread creation and management.

# Threads Synchronization and Communication

All threads in a process execute within a single address space and share resources. When threads share resources in an unsynchronized way, incorrect output can result from race conditions or thread scheduling anomalies. The DCE Threads Service provides the following facilities and routines to synchronize thread access to shared resources.

## Mutexes (Mutual Exclusion Objects)

Mutexes are used to synchronize access by multiple threads to a shared resource, allowing access by only one thread at a time. Routines for creating and managing mutexes are:

pthread_mutex_init(mutex,attr)
pthread_mutex_destroy(mutex)
pthread_mutex_lock(mutex)
pthread_mutex_trylock(mutex)
pthread_mutex_unlock(mutex)

## Condition Variables

Condition variables provide an explicit communication vehicle between threads. A condition variable is a shared resource, and requires a mutex to protect it. You use a condition variable to block one or more threads until some condition becomes true, then any or all of the blocked threads can be unblocked. Routines for creating and managing condition variables are:

pthread_cond_init(cond,attr)
pthread_cond_broadcast(cond)
pthread_cond_signal(cond)
pthread_cond_wait(cond,mutex)
pthread_cond_destroy(cond)

## Join Facility

The join facility is the simplest means of synchronizing threads, and uses neither shared resources or mutexes. The join facility causes the calling thread to wait until the specified thread finishes and returns a status value to the calling thread. Routines for joining and detaching threads are:

pthread_join(thread,status)
pthread_detach(thread)

Refer to the *OSF DCE Application Development Guide* (B3190-90036) and the *OSF DCE Application Development Reference* (B3190-90032) for detailed information on threads synchronization and communication.

## Threads Scheduling

HP 3000 Kernel Threads scheduling is handled through the dispatcher, therefore each thread is visible to and known by the kernel. Altering the scheduling of one or more threads in a task is accomplished with the same tools and methods used to alter the scheduling of any non-threaded task.

**NOTE** The HP 3000 Kernel Threads Service is a kernel based implementation of POSIX 1003.4a Draft 4 threads. Individual threads created within a given task may use the same processor at any given time; the threads are independently scheduled by the kernel. Therefore, a multi-threaded process can take advantage of the increased concurrency available on a multi-CPU machine.

## Writing Threaded Applications

The following are hints on writing multithreaded DCE applications:

- All DCE applications are multithreaded — When writing DCE applications, keep in mind, that the DCE runtime software is multithreaded and all DCE applications are multithreaded; even if the application code itself does not explicitly create threads.
- Using non-thread-safe libraries — When making calls to libraries you do not specifically know to be thread-safe, you must provide your own locking scheme to prevent multiple threads from executing the same library calls concurrently. While a given call may appear to be innocuous with respect to threads, it is very difficult to know exactly what interactions can occur within the library, or with other libraries. For example, suppose non-thread-safe routines 1 and 2 make a call to routine A (also non-thread-safe), if routines 1 and 2 use different mutexes to lock their calls to routine A, then routines 1 and 2 can both get into routine A at the same time (violating the programmer's attempt to make the calls thread-safe).
- Using fork() in a threaded application — fork() is not allowed from a threaded task.
- environ is a process-wide resource — Programmers must coordinate threads that use the putenv() and getenv() interfaces to change and read environ.
- Signal mask: a thread-specific resource — The signal mask is a thread-specific resource; therefore, if one thread manipulates the signal mask, it only affects signals that specific threads might be interested in (POSIX 1003.4a Draft 3 behavior).
- Handling synchronous terminating signals — The default behavior of OSF DCE 1.0.2 is to translate synchronous terminating signals into exceptions. If the exception is not caught, the thread that caused the exception is terminated. Any thread that goes through the terminate code causes the entire task to be terminated.
- Establish synchronous signal handlers using sigaction() — The MPE/iX POSIX C Library supports the following routines for setting up signal handlers:

  signal()
  sigaction()

Of these routines, only sigaction() is supported in a DCE application. It is used to establish handlers for synchronous signals on an individual thread basis only.

- Asynchronous signals — There is no supported mechanism for establishing signal handlers for asynchronous signals on MPE/iX.
- Cancelling threads blocked on a system call — The HP 3000 Kernel Threads Service provides a cancellation facility that enables one thread to terminate another. The cancelled thread normally terminates at a well-defined point. Terminating a thread that is blocked while executing system code is not possible on MPE/iX; only threads executing non-system code may be cancelled.
- Using waitpid() — The waitpid() routine allows the parent thread to specify which child it cares about by specifying its PID. This call only works for the initial thread; because children created by any thread within the task are considered children of the whole task.
- Using setjmp and longjmp — Do not use calls to setjmp and longjmp, these routines save and restore the signal mask and could inadvertently cause a signal that another

thread is waiting on to be masked. Instead, use _setjmp and _longjmp; these routines do not manipulate the signal mask.

When executing _longjmp be aware of the following:

— Ensure you are returning to a state saved within the context of the same thread.

— If you _longjmp over a **TRY** clause, an exception could try to _longjmp to a stack frame that no longer exists; and vice versa.

— Do not _longjmp out of a signal handler.

- Use **pthread_yield** to allow other threads processor time — If your application is running on a single-processor machine, and you want to permit other threads access to the processor, you can use **pthread_yield** to notify the scheduler that the current thread is willing to release the processor to other threads of the same or higher priority. If no threads of the same or higher priority are ready to execute, the thread continues.

  An example of the use of **pthread_yield** is to avoid spinning in a tight loop, such as:

  ```
  while (!flag);
  ```

  by using **pthread_yield** as:

  ```
  while (!flag) pthread_yield();
  ```

Use **pthread_yield** with caution; misuse can cause unnecessary context switching and increasing overhead with no increase in “fairness.” For example, it is counterproductive for a thread to yield while it has a needed resource locked.

## Writing Thread-Safe Code

The standard C/XL library is not completely thread safe on the HP 3000. Hewlett-Packard has provided a set of wrapper functions to intercept calls to the C library and make them thread safe. The wrapper definitions reside in the /usr/include/thdwrp.h file.

## Reentrant Interfaces

Many /lib/libc.a (POSIX C Library) routines return pointers to internal static data. This causes problems in a multithreaded program; while one thread tries to access the data another thread could be modifying it in some way.

The following are interfaces that should be called by multithreaded programs. These versions of the interfaces are different from the original versions.

The reentrant definitions currently defined in /lib/libc.a are:

| | | | |
|---|---|---|---|
| opendir_r | readdir_r | getgrgid_r | getgrnam_r |
| getpwnam_r | getwpuid_r | getlogin_r | |

In addition, the following are provided as part of /usr/lib/libdce.a on the HP 3000:

| | | | |
|---|---|---|---|
| asctime_r | crypt_r | ctime_r | ecvt_r |
| fcvt_r | gmtime_r | l64a_r | localtime_r |

There are four reentrant routines that are not defined in the DCE/3000 header files. To use them, add the following lines to the /usr/include/time.h file, just before the last line (#endif /* _TIME_INCLUDED */). For example:

```
#  ifdef _REENTRANT
       extern int asctime_r(const struct tm *, char *, int);
       extern int ctime_r(const time_t *, char *, int);
       extern int gmtime_r(const time_t *, struct tm *);
       extern int localtime_r(const time_t *, struct tm *);
#  endif /* _REENTRANT */stdio Interfaces
```

## stdio Interfaces

Some of the stdio interfaces (such as getchar() and putchar()) are available as functions within C/XL and as macros defined in stdio.h. The macro versions exist for performance reasons. Calls to the library implementation of these interfaces is intercepted by the thread wrapper functions, making them thread-safe; but the macros have not been made thread-safe. Since the thread-safe wrapper implementations should be used by default, the names of the macros have been changed by adding an _unlocked suffix (for example, **getchar_unlocked()**). This allows programmers to invoke the _unlocked macros and not pay the performance penalty of an extra function call and the cost of acquiring and releasing a lock.

Some of the _unlocked interfaces are outlined in POSIX 1003.4a, Draft 5. To support the use of the _unlocked interfaces, the functions flockfile() and funlockfile() are provided. These functions can be used to explicitly lock and unlock a file object. Therefore, exclusive access to a file for a series of _unlocked stdio calls is ensured, without having to lock and unlock the file on every call.

The additional stdio interfaces in thdwrp.h are:

| | |
|---|---|
| getc_unlocked | getchar_unlocked |
| putc_unlocked | putchar_unlocked |

The functions flockfile() and funlockfile() are implemented as part of /usr/lib/libdce.a.

## Debugging Threaded Applications

The following are hints for debugging threaded applications:

- Set breakpoints for global data bugs — To simplify debugging problems involving unexpected changes in global data, use **HPDEBUG** to set task-wide data breakpoints. Otherwise, make sure there is a breakpoint that every thread will hit. Even when you are single-stepping, another thread can intervene between source statements executed by the thread you are following. One way you can know that this is happening is if the intervening thread hits a breakpoint.

  You can prevent other threads from intervening while single stepping in **HPDEBUG**. This is done by setting the **HPDEBUG** environment variable **SS_TERM_KEEPLOCK**. Setting this variable causes you to hold the terminal semaphore for the current thread until a **CONTINUE** command is issued. Effectively, no other thread is able to obtain the terminal semaphore and interfere with your debug session. Another method is to create breakpoints within your task that have the “Stop-all-Threads” options. This option, when hit by any thread in the task, stops execution of all threads in the task until a **CONTINUE** command is issued. It is possible to create a deadlock situation when using **SS_TERM_KEEPLOCK** with “Stop-all-Threads” set in your task.

- Debugging a threaded server — Do not debug multiple threads in the server, debug one at a time. To do this, set a breakpoint at the procedure you want to catch and continue. Then make only one remote procedure call from the client; you will hit the breakpoint, step through until you locate the bug, then continue or quit.

- Debugging a threaded client — If the threads in a client do not interact, debug only one of them at a time.

# Process Management Commands

Process management (threads related) features of commands are listed here. For detailed information about the commands, refer to the *MPE/iX Commands Reference Manual* (32650-90006).

**SHOPROC**

- Using the ;PIN= parameter, pin.thread_id can be specified to denote a specific thread of the task.
- Using the ;FORMAT=SUMMARY option displays one line for each thread of the task. After the individual threads have been displayed, a composite or summary line is displayed containing the summation of the CPU time for all threads in the task, and the **QPRI** and **STATE** are blank. (This is done because, each thread in the task can have different values for the **QPRI** and **STATE** fields and there is no single value that can represent the task as a whole.)
- Using the pinspec parameter displays the process. The pinspec is a Process Identification Number (PIN). Any user can show processes matching their own user and account names; they belong to the user. A user with SM or OP capabilities can show any process (and its threads if any) on the system through its pin (or pin.thread_id). The System Manager can see system processes by specifying the SYSTEM option.

  If pinspec is 0, then the caller's pin is used.

  To reference a specific thread of a task, pinspec must be of the form pin.thread_id. Specifying pin.0 results in an error.

  **NOTREE** is the default for pinspec=[#p]pin target processes and can be overridden with the TREE option.
- Using the SUMMARY format displays a subset of a process' attributes. Included are the queue name, process priority, CPU time, execution state, associated JOB or SESSION number, PIN (indented to show tree structure), program name, threads related information (if any), and INFO=string (if any) or command step if the process is CI.PUB.SYS.

  SUMMARY is the default format for pinspec=[#p]pin.
- From the SHOWPROC display, the CPUTIME field is displayed in hh:mm:ss or m:ss.mls. A pair of asterisks (**) is displayed in the hours field if an overflow occurs. This is the summation of the CPUTIMEs for the individual threads in the task. For the individual threads, only the CPUTIME consumed by that thread is displayed.
- From the PIN(5) display, the PIN of an unthreaded task or the PIN of a threaded task is shown.
- From the PIN(9) display, the pin.thread_id of the individual threads in a threaded task are shown. Where, the pin.thread_id is abcde.fgh (abcde is the task PIN, and fgh is the thread_id). The SUMMARY format indents the PIN column by two spaces for each child process. The indentation is used to represent descendants.
- From the QPRI(5) display, a combination of the queue and priority is displayed (Qnnn [*]). Where, Q is a single character abbreviation of the processes scheduling queue, nnn is the processes priority, and * indicates a system process. When using the SUMMARY format the **QPRI** field is blank on the summary line of a threaded task.

- From the STATE(5) or STATE(7) display, the execution state of the process is indicated. **STATE** can be one of the following:

  | | |
  |---|---|
  | WAIT: | Generic process block, usually waiting for a message. |
  | BLKCB: | Blocked for control block. |
  | BLKMM: | Blocked for memory manager. |
  | READY: | Ready to execute or executing. |

Using the SUMMARY format, the **STATE** field is blank on the summary line of a threaded task.

**ALTPROC**

- Using the ;PIN= parameter, pin.thread_id can be specified to denote a specific thread of the task.
- Specify pin.thread_id to alter the attributes of a single thread.
- Using the pinspec parameter displays the processes whose attributes are to be altered. Any user can show processes matching their own user and account names; they belong to the user. A user with SM or OP capabilities can alter any process (and its threads if any) on the system through its pin (or pin.thread_id). The System Manager can see system processes by specifying the SYSTEM option.

  If pinspec is 0, then the caller's pin is used.

  To reference a specific thread of a task, pinspec must be of the form pin.thread_id. Specifying pin.0 results in an error.

  **NOTREE** is the default for pinspec=[#p]pin target processes and can be overridden with the TREE option.

# Process Management Intrinsics

Process management (threads related) features of intrinsics are listed here. For detailed information about the intrinsics, refer to the *MPE/iX Intrinsics Reference Manual* (32650-90013).

**PROCINFO**

The PROCINFO intrinsic returns threads related information to the caller. Four item numbers do this:

- Item#=13 (threaded task option (I32)) — returns an integer that can have one of the following values:

  | | |
  |---|---|
  | 0 | This task was never multi-threaded during its life-span. |
  | 1 | This PIN was multi-threaded at some point during its existence (for example, it was created by the **RUN** command or CREATEPROCESS intrinsic and has executed at least one PTHREAD intrinsic. |
  | 2 | The task is currently multi-threaded. |

  The user must have PM capability.

- Item#=14 (thread type option (I32)) — returns an integer that can have the following values:

  | | |
  |---|---|
  | 0 | This PIN was never multi-threaded. |
  | 1 | The PIN passed is that of the initial thread (for example, a process created by the **RUN** command or CREATEPROCESS intrinsic that has executed at least one PTHREAD intrinsic). |
  | 2 | The PIN passed is that of a secondary thread of the task. |

  The user must have PM capability.

- Item#=15 (number of threads option (I32)) — returns an integer that can have the following values:

  | | |
  |---|---|
  | 0 | This PIN was never multi-threaded during its life-span. |
  | 1 | This PIN was multi-threaded at some point during its existence (for example, it was created by the **RUN** command or CREATEPROCESS intrinsic and has executed at least one PTHREAD intrinsic. |
  | n | This PIN is part of a threaded task and n is the number of threads currently associated with the task. |

  The user must have PM capability.

- Item#=16 (list of thread PINs option (16-bit signed integer array)) - the user must pass in an array large enough to hold the PINs of all threads associated with the task. The first element of the array must contain the array size; PROCINFO fills the array starting from the second element. If the array size is not large enough, PROCINFO fills the available space and returns an error indicating that the array size was insufficient. The last element of the array will be a zero. The user must have PM capability.

**SUSPEND and ACTIVATE**

If a thread invokes SUSPEND (or ACTIVATE) only that thread is suspended. Other restrictions for suspending a thread are:

- Child waits — A thread is only be permitted to wait for the children it created. An attempt to wait on a child created by another thread results in an error.
- Father waits — Only the initial thread can issue a father wait. An error is returned to secondary threads issuing a father wait.

**FATHER**

All threads have the same father, namely the father of the initial thread. If a secondary thread creates a child process, the father of the child process is the initial thread of the task.

**GETPRIORITY**

A threaded process can change its own priority. However, an initial thread or a secondary thread cannot have its priority changed by another process.

**GETPROCINFO**

This works the same with any threaded process.

**KILL**

The children of a task can only be killed by the initial thread. A KILL issued by a secondary thread returns an error indicating that the thread is not the father of the child.

**PROCTIME**

This routine returns the CPU time that the thread has accumulated.

**SENDMAIL and RECEIVEMAIL**

An error is returned under the following conditions:

- If called from a secondary thread.
- If calling SENDMAIL and sending to a child that is a secondary thread.
- If calling RECEIVEMAIL and receiving from a child that is a secondary thread.
- If calling SENDMAIL and sending to the father and the father is a thread (either initial or secondary).
- If calling RECEIVEMAIL and receiving from the father and the father is a thread (either initial or secondary).

**TERMINATE and QUIT**

If a thread calls TERMINATE or QUIT, the task, including all of its threads, is terminated.

# Changes to AIF Routines

The following AIF routine item numbers are associated with kernel threads:

**AIFPROCGET and AIFPROCPUT**

The following are the associated item numbers:

| Item# | Name (Type) and Description |
|---|---|
| 2137 | Thread Type (I32) — Indicates what type of thread this process is. Valid values are:<br>0 — Regular process (not a thread).<br>1 — Initial thread (process created by the **RUN** command or CREATEPROCESS that has executed at least one PTHREAD intrinsic).<br>2 — Secondary thread (process created by the HPTHDCREATE intrinsic). |
| 2138 | Initial Thread Pin (I32) — Returns the PIN of the initial thread for this process. If the process is not part of a threaded task, then a PIN of 0 is returned. If the process is an initial thread, then its own PIN is returned. |
| 2139 | Initial Thread PID (REC) — Returns the PID of the initial thread for this process. If the process is not part of a threaded task, then a PID of 0 is returned. If the process is an initial thread, then its own PID is returned.<br>Record type: longint_type (Refer to the *AIF Reference Manual*) |
| 2140 | List of Secondary Thread PIDs (REC) — Returns of list of PIDs of all the secondary threads. The first word must hold the size, in longwords, of the rest of the buffer area. The first word, upon return, specifies the number of PIDs returned.<br>This item is relevant only if the process is an initial thread or a secondary thread. If a regular process is specified, then no PIDs are returned and a value of 0 is returned in the first word.<br>Record type: I64rec_type (Refer to the *AIF Reference Manual*). |
| 2141 | TIN (I32) — Returns the Thread Identification Number. Each thread must have a unique TIN. For a regular process (non-thread) the value 0 is returned. |

**AIFSYSWIDEGET** Following is the associated item number:

| Item# | Name (Type) and Description |
|---|---|
| 2137 | Thread Type (I32A) — Passing this criteria returns the PIDs of processes with the thread type equal to the specified criteria value. Valid values for this item are: |
| | 0 — Regular process (not a thread). |
| | 1 — Initial thread (process created by the **RUN** command or CREATEPROCESS that has executed at least one PTHREAD intrinsic). |
| | 2 — Secondary thread (process created by the HPTHDCREATE intrinsic). |

# Input Reference Parameter Protection for Intrinsics

When an intrinsic accepts a reference parameter, the data within that parameter could be changed by another thread if the data does not lie on the calling thread's stack. If it appeared that a corrupt input reference parameter could either cause the system to abort or corrupt system data structures, protection was added to that intrinsic.

The following list contains intrinsics that make input reference parameter protection difficult. Programmers need to provide their own scheme for protecting the input reference parameter data during the execution of these intrinsics.

**FDEVICECONTROL** The *LENGTH* parameter given to the intrinsic must exactly describe the length of the *BUFFER* parameter. If *LENGTH* is either longer or shorter than the actual length of the *BUFFER* parameter, unexpected results could occur.

**FCONTROL** There is no input reference parameter protection provided for the *PARAM* parameter.

**AIF Intrinsics** There is no input reference parameter protection provided for any of the *AIF* parameters.

**POSIX Intrinsics** The following POSIX intrinsics were found to have input reference parameters that could be corrupted when used in a threaded environment.

| POSIX Intrinsic | Input Reference | Parameter Name |
|---|---|---|
| chmod | path | |
| closedir | dirp | |
| mkdir | path | |
| opendir | dirname | |
| opendir_r | dirname | |
| readdir | dirp | |
| readdir_r | dirp | |
| rewinddir | dirp | |
| rmdir | path | |
| unlink | path | |
| creat | path | |
| open | path | |
| getenv | name | |
| ioctl | arg (and all areas pointed to by arg) | |

**MKS Routines**

Some routines from MKS available for POSIX are:

| | | | |
|---|---|---|---|
| confstr | fnmatch | getopt | glob |
| popen | regerror | regexec | sysconf |
| system | wordexp | | |

It is up to the caller to provide the necessary protection scheme for input reference parameters used by the MKS routines.

## File Access From Threads

The following is a high-level overview of the file system functionality available:

- The file system supports thread sharable NM disk files, including user mapped files and spoolfiles. CM files (tapes, cir, rio, CM KSAM) are thread private (Thread-Non-Sharable). Thread-Semi-Sharable terminals and printers are supported. Thread-Semi-Sharable means that some, but not all, intrinsics can be called from all threads for these file types.
- **PRINT** and **READX** can be used by any thread if $STDIN or $STDLIST is an AVESTA or Virtual Terminal. They are only supported for the initial thread, if the terminal is a **TMUX** Terminal. **READ** is supported for the initial thread only for all terminal types.
- Only the initial thread can open Thread-Non-Sharable or Thread-Semi-Sharable files as system files. Only the initial thread can re-open system files if the system files are printers or terminals. However, **HPFDUPLICATE** allows each thread to get their own file descriptor for a system file.
- **FCHECK** (filenum) returns the last error that the calling thread incurred on the file if that error is also the last file system error for the calling thread.
- **FLOCK** behaves as an advisory lock outside task boundaries; if one thread in a task has a file FLOCKed, all other threads block if they call a file system intrinsic on the file.
- For **IOWAIT**, only the thread that opened the file for **NOWAIT IO** can call **IOWAIT** for that file, and only the thread that opened the file with **NOWAIT** can call any **IO** intrinsics on the file.

## GlanceXL

GlanceXL displays a task's thread information and child information separately. The thread information is not formatted like that found in **SHOWPROC**. Each thread looks like an individual process, having it's pin number (used by the dispatcher) displayed, as opposed to it's thread number within the task.

# XL.PUB.SYS

In a threaded environment, any XL module that used global or static variables has the potential to have these data areas corrupted. The following table lists the modules that appear to be safe to use in a threaded environment. However, until all modules are explicitly thread-safe, it is recommended that you provide your own locking scheme to prevent multiple calls to all entry points.

**Table 5-1 Thread-Safe XL.PUB.SYS Modules**

| Module Name | Description | Module Name | Description |
|---|---|---|---|
| STIS209S | VPLUS/V | SENTRYTI | TURBOIMAGE/XL |
| HP31900 | MPE/iX | HP36961 | NCS RUNTIME |
| PSICOMN | PSI COMMON | FMT | IND FORMATTER |
| LANCELOT | NIO LAN DRIVER | LSS | IND LSS |
| SOCKET | NET SOCKETS | S01STLIB | TOOLSET LIBRARY |
| NMEVNT | SYSMGR NMEVENT | S25S391C | TURBOIMAGE/XL |
| U_QFABS | CORE LIBRARIES | S29S391C | TURBOIMAGE/XL |
| ACTUTILS | IMAGE/SQL | HP32007 | BSC LINK/XL |
| HP32015 | HP LAN MANAGER | DBCORE.P | ALLBASE/XL HP SQL |
| HP30293 | SNA IMF/XL | HP30294 | LU 6.2 API/XL |
| HP32589 | HOST DIAGNOSTIC | HP36936 | HP SYSTEM MANAGER |
| HP36957 | FTP/XL | STEALTH | FDDI LAN DRIVER |
| GALAHAD | TOKEN RING DRIVER | AHPDINT | INTRINSICS DRIVER |
| PSILAPB | PSI LAPB DRIVER | SNMP | HP SNMP/XL AGENT |
| PSISDLC | PSI SDLC DRIVER | HPSQL2 | ALLBASE/XL HP SQL |
| HPSQL3 | ALLBASE/XL HP SQL | HP32209 | VPLUS |
| HPSQL5 | ALLBASE/XL HP SQL | HPSQL8 | ALLBASE/XL HP SQL |

## PTHREAD Intrinsics

The following are the PTHREAD intrinsics supported for DCE/3000. The PTHREAD procedure and type declarations are located in the PTHREADH.THREADS file.

```
int
pthread_cond_broadcast (cond)
  pthread_cond_t *cond;
int
pthread_cond_destroy (cond)
  pthread_cond_t *cond;
int
pthread_cond_init (cond, attr)
  pthread_cond_t *cond;
  pthread_condattr_t attr;
int
pthread_cond_signal (cond)
  pthread_cond_t *cond;
int
pthread_cond_timedwait (cond, mutex, abstime)
  pthread_cond_t *cond;
  pthread_mutex_t *mutex;
  struct timespec *abstime;  /* version 5 syntax */
int
pthread_cond_wait (cond, mutex)
  pthread_cond_t *cond;
  pthread_mutex_t *mutex;
int
pthread_mutex_destroy (mutex)
  pthread_mutex_t *mutex;
int
pthread_mutex_init (mutex, attr)
  pthread_mutex_t *mutex;
  pthread_mutexattr_t attr;
int
pthread_mutex_lock (mutex)
  pthread_mutex_t *mutex;
int
```

```
pthread_mutex_trylock (mutex) /* returns 0 if owned */
  pthread_mutex_t *mutex      /* by current thread  */
                              /* (ver 5) versus     */
                              /* -1 (ver 3)         */
int
pthread_mutex_unlock (mutex)
  pthread_mutex_t *mutex;
int
pthread_cancel (thread)
  pthread_thread_t thread;
int
pthread_setasynccancel (state) /* May only set async */
                               /* cancel off!!       */
   int  state;                 /* Async Cancels not  */
                               /* supported.         */
int
pthread_setcancel (state)
   int  state;
void
pthread_testcancel ()
int
pthread_keycreate (key, destructor) /* version 5 syntax.*/
  pthread_key_t *key;               /* The destructor is*/
  void (*destructor) ()             /* now called when  */
                                    /* thread exits!    */
int
pthread_getspecific (key, value) /* version 4 semantics */
  pthread_key_t key;
  void **value;
int
pthread_setspecific (key, value) /* version 4 semantics */
  pthread_key_t key;
  void *value;
int
pthread_once (once_block, init_routine)
  pthread_once_t *once_block;
  void (*init_routine)();
pthread_thread_t
```

```
pthread_self ()
void
pthread_yield ()
int
pthread_attr_create (attr)
  pthread_attr_t *attr;
int
pthread_attr_delete (attr)
  pthread_attr_t *attr;
int
pthread_attr_setstacksize (attr, stacksize)
  pthread_attr_t *attr;
  long stacksize;
int
pthread_create (thread, attr, start_routine, arg)
  pthread_thread_t  *thread;
  pthread_attr_t attr; /* attr IGNORED! */
  pthread_
  void *(*start_routine) ();
  void *arg;
int
pthread_join (thread, exit_status)
  pthread_thread_t  thread;
  void **exit_status;
int
pthread_detach (thread)
  pthread_thread_t  *thread;
pthread_exit (exit_status)
   void *exit_status;
int
pthread_get_expiration_np (delta, abstime)
  struct timespec    *delta;
  struct timespec    *abstime;
int
pthread_delay_np (interval)
  struct timespec  *interval;
```

The following macros are defined in pthread.h to register and unregister per-thread cleanup handlers:

```
pthread_cleanup_push(routine_parm, arg_parm)
pthread_cleanup_pop(exception)
```