Performance Characteristics of Key Functions Using IBM DCE for OS/2

June, 5 1996


```
Debra A. Berding
Cindy Corn
Tim Li
Ray Pekowski
Bob Santeford
```


Contents

Abstract

This article describes some basic performance characteristics of IBM(*) Distributed Computing Environment (DCE) for OS/2(*). It gives insights into the performance of different core components, in particular Remote Procedure Call, Cell Directory Services and Security Services, through various benchmarks. The focus is on a set of the most important basic functions of DCE. Some tips and helpful hints are given for performance improvements. The information in this article may help users to configure and use DCE for OS/2 more efficiently.

Trademarks

IBM, PS/2, OS/2 denoted by an asterisk (*) in this article, are registered trademarks of International Business Machines Corporation.

Open Software Foundation, OSF, denoted by (**) in this article are registered trademarks of the Open Software Foundation, Inc.

Introduction

IBM(*) Distributed Computing Environment (DCE) for OS/2 is a fundamental building block for distributed computing in an open systems environment. It provides services and tools that support the creation, use and maintenance of distributed application in the PC-LAN environment. The current implementation is based on the Open Software Foundation (OSF)(**) DCE 1.1. This article addresses some performance characteristics of DCE for OS/2 with key basic functions in various core components of DCE, in particular:

- Remote Procedure Call (RPC)

  Extends the typical procedure call model by supporting direct calls to procedures on remote systems.

- Cell Directory Services (CDS)

  Defines a naming model that allows users to identify resources by name without knowing their location.

- Security Services

  Provides encryption and authentication that ensures the privacy and authenticity of client/server transactions.

- Threads

  Supports the creation, management, and synchronization of multiple threads of control within a single process.

In this article, for each of these components, a simple tutorial is given first, followed by performance information, and finally, tips or helpful hints are given. We hope that this information may help users to configure and use DCE for OS/2 more efficiently.

System Configuration

The hardware configuration used in this article was as follows:

- DCE Server - 90 Mhz Pentium with 32 MB RAM and 256K L2 cache.
- DCE Client - 90 Mhz Pentium with 32 MB RAM and 256K L2 cache.
- Connected by IBM Token-Ring 16/4 Adapter/A at 16 Mbps

The software configuration used in the study was:

- IBM OS/2 Warp, version 3.0
- IBM Distributed Computing Environment for OS/2 version 2.1

Remote Procedure Call (RPC)

The distributed communications model used by DCE is the remote procedure call. In this model, the client makes what looks like a procedure call. The procedure call is translated and communicated to a server routine. The server routine believes it has been called locally. When the server returns, the results are communicated back to the client.

The DCE component that implements this model is the Remote Procedure Call or RPC. All other components of DCE use RPC for their communication needs. Therefore, RPC plays an important role in DCE performance.

The client and server programs are usually on two separate machines with possibly differing architectures. RPC runtime takes care of marshalling (copying input and output parameters to communication buffers), managing the communications link, and translating any architectural differences (such as byte reversal) between the client and server machines.

OS/2 DCE RPC operates over the Transmission Control Protocol/Internet Protocol (TCP/IP) communications service and supports both of its underlying transports: Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). TCP provides a reliable connection oriented service, while UDP provides an unreliable datagram service. Despite the differences between these two transports, RPC provides the same level of function (same semantics) over each of them. Any lack of function in a transport (such as lack of reliability) is made up for with function in RPC.

RPC is integrated with other DCE components and uses them to provide services such as finding servers (through the directory services) and providing security (through authentication and encryption) over the communication network. RPC uses the DCE Threads component to allow multiple concurrent calls to the same server routine. The number of concurrent calls is selectable by the server program and can be one. An alternative way of making the same server routine available for multiple concurrent calls is to implement multiple server processes. RPC gives you the flexibility to choose one or a combination of both these techniques.

DCE RPC supports a rich set of data types enabling you to choose a natural set of parameters for your procedure or interface definition.

Benchmarks

The following benchmarks are used to evaluate RPC performance:

- RPC pipe transfers some specified number of bytes from a client to a server using the RPC pipe data type. It is used to compare the performance of different data types.
- Security RPC has a null case, (Null RPC has no parameters or return value. Its purpose is to show the base line performance cost of an RPC), and a data transfer case (client to server), with a specified level of security. It is used to observe the performance cost of security.

  The level of security is selected from the following possibilities:

  none
  : No security

  connect
  : Perform protection only when the client establishes a relationship with the server.

  call
  : Perform protection only at the beginning of each remote procedure call when the server receives the request.

  packet
  : Ensure that all data received is from the expected client or server.

  packet integrity
  : Ensure and verify that none of the data transferred between client and server has been modified.

  packet privacy
  : Perform protection as specified by all of the previous levels and also encrypt each remote procedure call argument value.

Performance Data

The measurements in the subsequent sections are for the steady state environment. Load and initialization times are not included.

Base RPC with TCP versus UDP

A comparison of the base cost of doing RPC for the two TCP/IP protocols supported, UDP and TCP, is shown in Figure 1. The security benchmarks with security set to none are used. The TCP for RPC performs slightly better than UDP for data sizes of 8 KB and greater. Although the 32 KB and 100 KB data points have not been included on Figure 1 (so the smaller data points can be distinguished from each other), the line has been drawn extending from the 16 KB data point to where the 32 KB and 100 KB data points would be if the chart were larger. The full set of response times is in Appendix A.

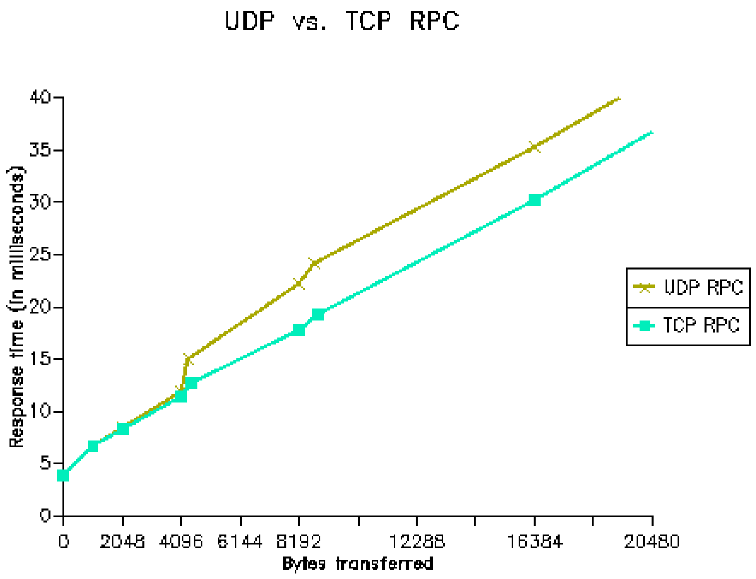

Figure 1: Base RPC

A common characteristic of communications protocols is that response time increases linearly with the data length from zero to some point at which there is a sharp increase in response time. This point approximates the block or packet size -- the size at which the protocol breaks up the data. Figure 1 implies the RPC packet size is between 4200 and 4250 bytes and 8450 and 8500 bytes for UDP RPC. The RPC packet size is between and 4150 and 4350 and 8500 and 8650 for TCP RPC.

Pipes Versus Arrays

Are all data types treated alike from a performance perspective? Two data types are examined here, arrays and pipes. One difference between the two is the interface used to program to them. They both start out as the standard procedure call model, but pipes have an additional callback mechanism for getting at the pipe data. The callback consists of one or more calls from the RPC runtime back to an application routine. Through these callbacks, the RPC runtime requests the addresses and sizes of the data to be transferred. A possible disadvantage of the callback model is that it might be more difficult to program to -- it breaks up the sequential programming flow of the client application.

Pipes have a performance advantage because the RPC data does not have to be immediately available at the start of the RPC. The data can be sent via the callback mechanism as it becomes available. A new RPC does not have to be initiated for each amount of data to be sent.

Figure 2 compares throughput for pipes and RPCs for various data sizes. Data size refers to the total amount of data passed to RPC. The throughput discussed here is for single client and single server. When measuring this type of throughput, the RPC internal blocking size is immaterial. If a large enough amount of data is transferred, the maximum throughput rate for pipes is reached.

J

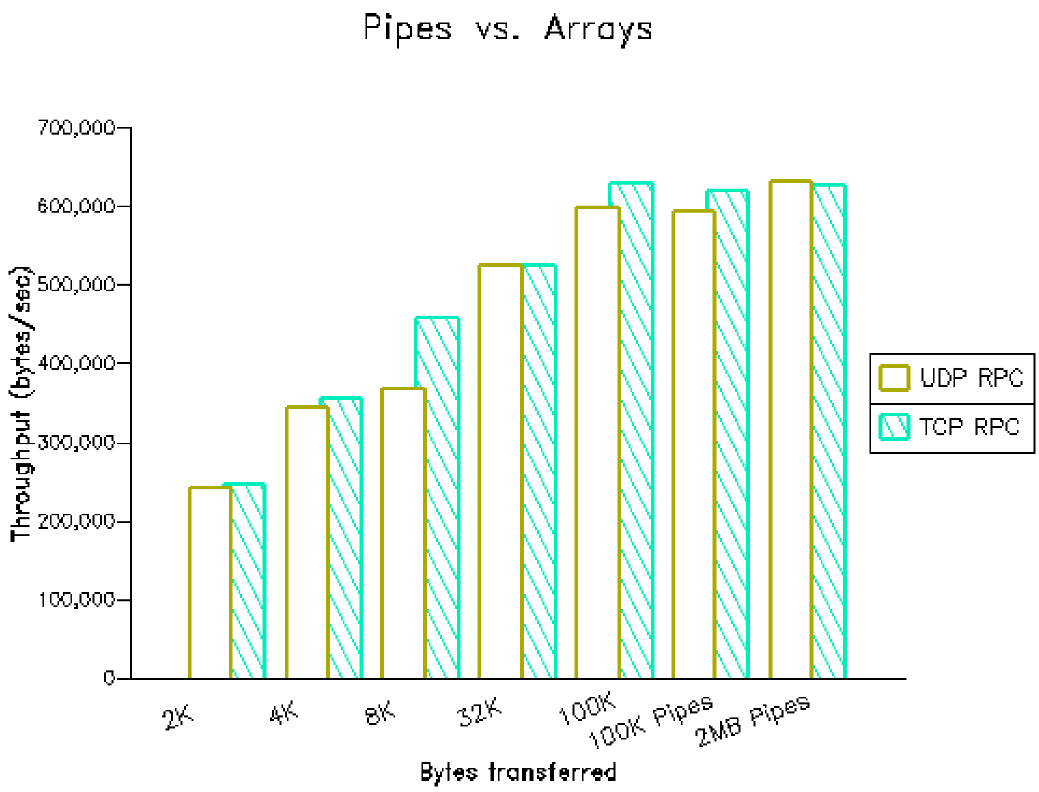

Figure 2: Pipes Versus Arrays

Note that 2MB pipes achieves an equal to or slightly better level of performance than the 100K variable length array case, but 100K pipes are slightly slower than the 100K variable length array case.

Why would you choose pipes over arrays? If the data is not immediately available, if obtaining the data can be overlapped with sending it (or receiving the data can be overlapped with processing it on the server), and if the time to obtain the data is on the same order as the time to send it, we recommend that you use pipes.

Local versus Remote

When the client and server are on the same machine, local RPCs are optimized (by using Unix domain sockets).

What is the performance difference of an RPC to the same (local) machine versus to another (remote) machine? Figure 3 compares local RPC to remote RPC for various data transfer sizes. Local RPC is faster than remote RPC for data sizes of null to 100 KB for both TCP and UDP. However, Local TCP is slightly faster than local UDP for data sizes of 8 KB and greater.

]

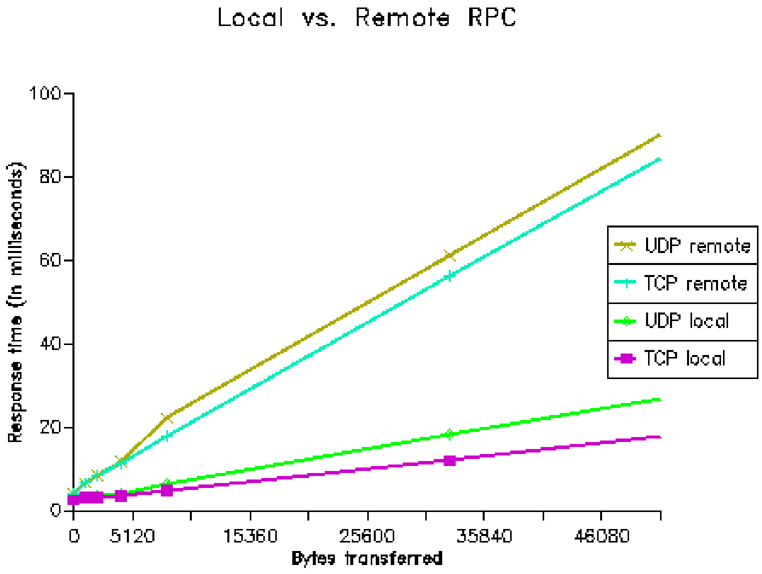

Figure 3: Local Versus Remote RPC

RPC with Security

What is the performance cost of doing a secure RPC? Figure 4 compares RPC at various levels of security. We see that the security levels of none, connect, call, and packet perform similarly, but a large performance degradation exists for packet integrity and an even larger one for packet privacy. The overhead for security -- the additional time to provide the service -- is about the same for TCP as UDP.

]

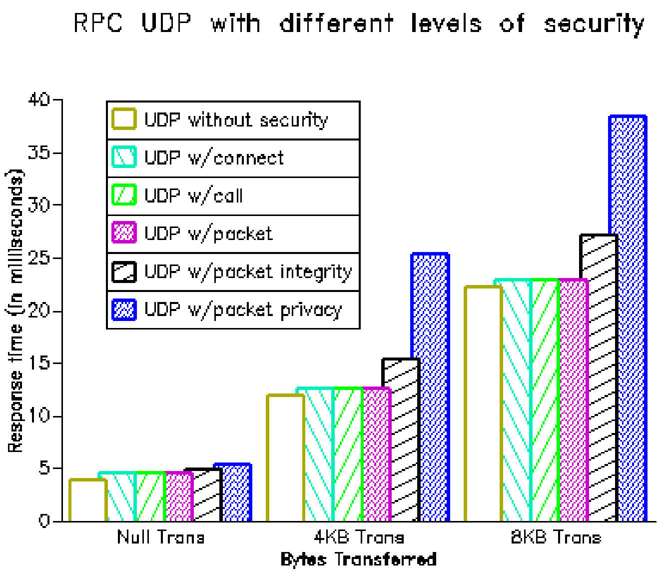

Figure 4: RPC With Security (UDP)

Tips and Helpful Hints

- Use TCP RPC for data sizes larger than 8 KB. Use UDP RPC for pipes with data sizes equal to or larger than 2 MB.
- Use RPC pipes if the data to be transferred is not immediately available and obtaining it can be overlapped with sending it.
- Where possible, minimize the use of RPCs that have security levels of packet integrity and packet privacy (encryption). Use the minimum level necessary for your application.

Cell Directory Service (CDS)

Cell Definition

A cell consists of a collection of users, computers and resources that share a common set of DCE services. The cell is the primary unit of administration in DCE and is usually defined so that control centers around a common purpose, such as an organization within a company. At a minimum, the cell configuration must contain one Security Server and one Cell Directory Server. These servers may be contained in one or more machines. DCE currently provides support for multiple cell networks through a Global Directory Agent in the cell. The network topology may be quite varied within a single cell, ranging from a small Local Area Network (LAN) to an extensive set of LANs and Wide Area Networks (WANs).

Cell Directory Service Overview

The Cell Directory service looks up and manages names of resources in a cell. The names are specified in a hierarchical arrangement that defines the cell's namespace. A CDS directory appears very much like the familiar file system directory, particularly for UNIX users or like the OS/2 and DOS path name without the drive letter and ":" prefix. A directory resides on a CDS Server in a Clearinghouse. The clearinghouse can be thought of as a data base where the directory information is stored.

There are essentially three kinds of entries in a CDS directory: a directory entry (also called a child pointer), an object entry and a soft link. The directory entry defines the hierarchical namespace with relative names separated by a "/" delimiter. A directory entry connects upper levels of a directory to another directory immediately beneath it in the cell namespace. An object entry represents a resource or entity of some sort. It is the end point, or leaf node, of the directory. A soft link is a pointer that provides an alternative name (alias) for a directory, an object or another soft link. Another kind of end point is a junction. A junction provides a connection to special services, such as Security and the Distributed File System, independent of the directory service. It is an object entry for the servers providing that service.

The CDS Server responds to namespace requests from client applications, which includes the other DCE components: RPC, Security, DFS and DTS. DCE applications use CDS for locating servers and clients. Every client has a CDS Advertiser and at least one CDS clerk. A clerk initiates CDS operations and stores the directory data it retrieves in a local memory cache. The CDS Advertiser starts clerks as required, locates the CDS Servers in a cell and manages the cache. CDS server locations are cached for later use. Caching allows fast retrieval of previously acquired directory information avoiding repeated requests to the server. Periodically the cache is written to disk so that it can survive system crashes and re-boots. The cache is also written to disk when the CDS Advertiser is stopped. (A slim client configuration is also available which allows the CDS clerk to perform the responsibilities of both the CDS clerk and the CDS Advertiser.)

CDS Performance Measurements

The CDS Control Program (CDSCP) can be used by a cell administrator, or other users with the appropriate permissions, to manipulate and manage CDS directories. This tool is the basis for the CDS performance measurements. The measurement cases are defined as follows:

- Specific commands have been timed by repeated execution in a batch program. The times obtained are fairly representative of the response times obtained for a manual entry from the keyboard.
- The CDSCP commands selected for measurement are SHOW and LIST. LIST simply lists the entries at a particular directory level. SHOW displays detailed information about the entity and its attributes.
- The commands are timed for both directory entries and for object entries.
- The commands are executed at high levels of the directory hierarchy.

Figure 5 shows the base set of measurements of the aforementioned CDSCP commands. List and Show perform similarly for both directories and objects.

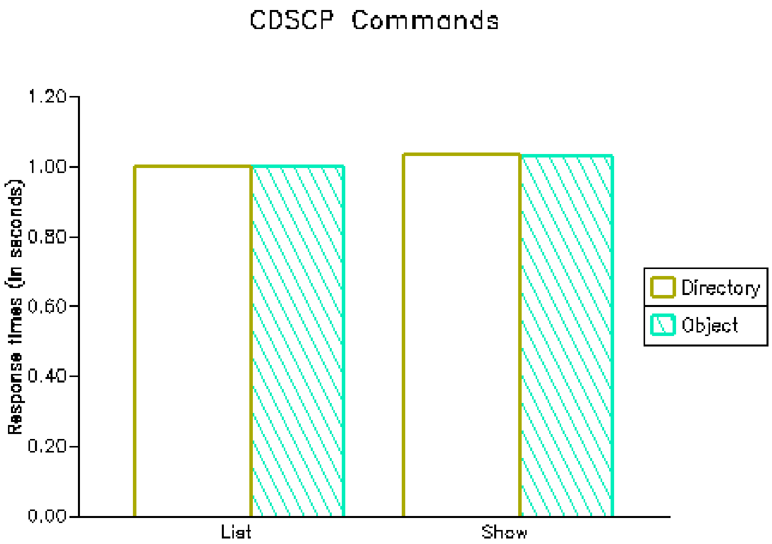

Figure 5: CDS Control Program Base Commands

The DCE control program, DCECP, can also be used to manipulate and manage CDS directories. The information contained in the output of the DCECP Directory commands is the same as CDSCP, but it is formatted differently. DCECP is the new single control program which contains all of the function of the previous control programs (rpccp, acl_edit, cdscp and rgy_edit) plus more function. Figure 6 compares the performance of CDSCP and DCECP commands. DCECP commands are slightly slower than their CDSCP counterparts. This is because DCECP has more overhead since it provides other functions such as shell command ability and TCL interpretation. Also, DCECP is larger in size and uses more DLLs, so the load time is longer than for CDSCP.

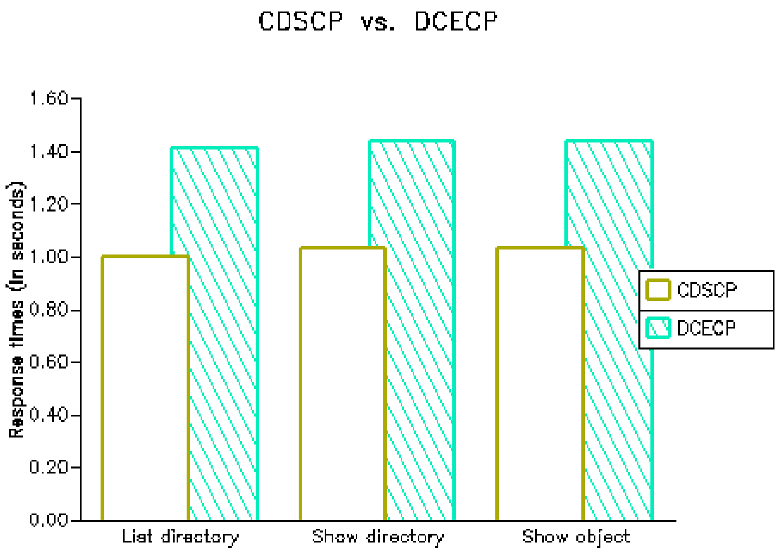

Figure 6: CDSCP vs. DCECP

Tips and Techniques

Although the CDS directory looks like a normal file system directory, it is much more than that and consequently the response time of directory lookups is significantly greater in CDS. Thus, CDS should be used for its primary purpose of providing resource location information and should not generally be used for data that is more suitable for a file or a data base. In particular, CDS is designed to be used for frequent reading of information that does not change often, so that the client-side caching capability is exploited. It is not designed to be used by applications that need to write data frequently. CDS has no mechanism for invalidating client caches. Applications specify how old the data may be. DFS uses a much stronger caching model with callbacks for cache invalidation and should be used by applications that need to write data frequently.

Security Services

The DCE Security Service enables clients and servers to prove their identities to each other, offers integrity and privacy of communications, and supports controlled access to resources.

The DCE Security Service maintains its own namespace to ensure that the DCE cell remains secure. Clients of the Security Service query CDS for binding information that enables them to find the Security Server.

This article will concentrate on the following key security commands:

DCELOGIN

The dcelogin command authenticates the user to the Security Service by means of the user’s password, thereby establishing an authenticated network identity. The Security Server knows that the name entered on the dcelogin command is the name of a principal (user) which is registered in the security part of the namespace.

DCELOGIN AUTOMATION

The "dcelogin" command acquires a DCE identity and creates a subshell in which that DCE identity can be used when executing DCE programs. In order to automate the performance measurements for the login function, a less well-known feature "dcelogne" is used. "dcelogne" is not shipped with the product, but can be created by copying the "dcelogin.exe" file to "dcelogne.exe". The code functions differently depending on the program name invoked.

The "dcelogne" command does not spawn a new shell, as does dcelogin, but results in the creation of a new credentials file. "dcelogne" emits the name of this file and then returns. However, the environment variable (KRB5CCNAME) which is automatically set by "dcelogin" must be manually set to the output of dcelogne. (Note that this technique will not work with DFS.)

DCECP Security commands

The DCECP Security commands allow the user to edit the registry database which contains information on principals, groups, organizations, accounts, and administrative policies. The DCECP Security commands also allow the user to manage access control lists (ACLs) for all namespace objects. An ACL contains individual ACL entries that specify the permissions granted a user to the object with which the ACL is associated. The ACL entries collectively determine which principals can use the object and what operations they are allowed to perform on it. Since each ACL Manager defines the permission tokens and meaning appropriate for the objects it controls, the actual tokens and their meanings vary.

PERFORMANCE OF KEY SECURITY COMMANDS

DCECP Security commands:

Figure 7 shows performance measurements for some common Administrative security operations using the DCE control program. The DCECP "Show" command will be used by Administrators and other users to view accounts, principals, organizations, and groups.

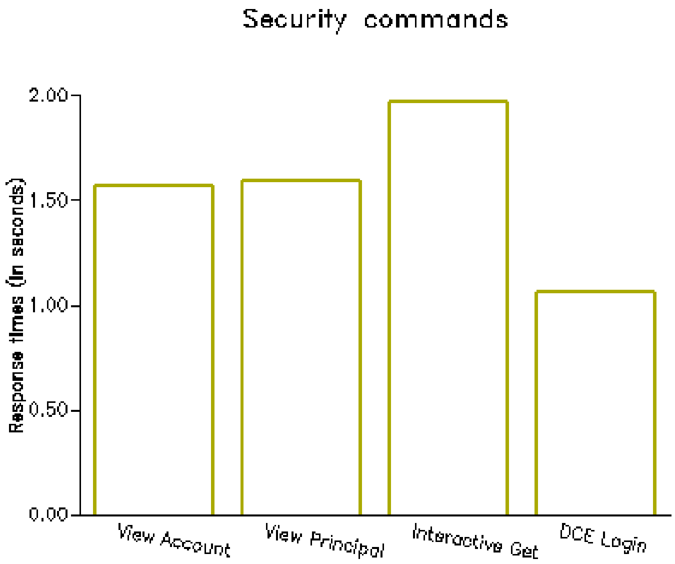

Figure 7: DCECP Security Operations

TIPS AND HELPFUL HINTS:

- Use "dcelogne" to automate the "dcelogin" function.

Threads

DCE Threads is a facility that provides multiple concurrent paths of execution within a process, as opposed to a single path of execution available in the traditional UNIX model of programming. DCE Threads is based upon POSIX draft standard 1003.4a. Other components of DCE use Threads because of its functionality and portability.

OS/2 has its own threading facility. OS/2's DCE implementation is a mapping of the DCE Threads APIs to the corresponding OS/2 APIs.

DCE Threads APIs and OS/2 threads APIs can coexist within the same process, but they cannot interoperate. That is, you can use both APIs within the same process, or even the same thread; but you cannot, for example, use one set of APIs to wake up a thread that is blocked in the other set of APIs.

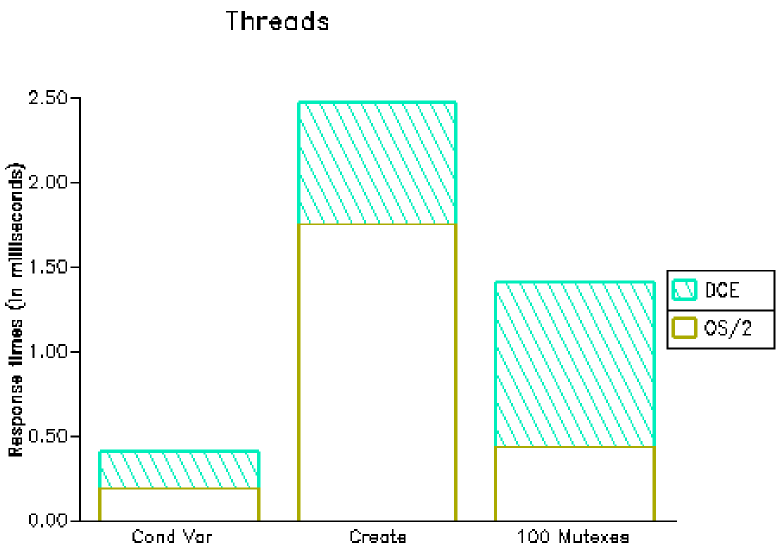

Figure 8: Threads

From a performance perspective, DCE Threads APIs are a layer on top of OS/2's threads APIs, and so must be slower. Because an application's use of threads is usually only a small portion of its total, the overhead associated with using DCE Threads is likely to be insigificant. Even so, we generally recommend that, if you are not porting to other platforms, you should use the OS/2 Threads APIs to achieve better performance.

The Condition Variable test case measures the time to transfer control back and forth between two threads using condition variables. (Two threads switches, two pthread_cond_wait and two pthread_cond_signal are measured.) The Mutex test case measures the time to lock and unlock 100 mutexes. The Create test case measures the time to create a thread, join a thread and then detach it. Figure 8 shows the performance of the threads test cases, splitting out the OS/2 and DCE threads components of the overall response time.

Summary

This article has addressed the following key points:

- For RPC
  - The performance varies by many factors, such as: size of data transfer, security levels, transport used, pipes vs arrays, local vs remote and hardware used.
  - Use TCP RPC for data sizes larger than 8 KB. Use UDP RPC for pipes with data sizes equal to or larger than 2 MB.
  - Use RPC pipes if data to be transferred is not immediately available and obtaining it can be overlapped with sending.
  - Where possible, minimize the use of RPC with security levels of packet integrity and packet

privacy.

- For CDS
    - CDS should not be used for data that is more suitable for a file or a database.
- For Security Services
    - Use "dcelogne" to automate the "dcelogin" function.
- For threads
    - OS/2 threads are preferred for performance, DCE threads are preferred for portability.

Appendix A: Response Time Data

The following are the actual data measurements. All measurements are in milliseconds.

```
RPC...UDP...no_security
NULL_TRANSFER......          3.847
1_KB_TRANSFER......          6.635
2_KB_TRANSFER......          8.413
4_KB_TRANSFER......         11.888
4250_B_TRANSFER             15.062
8_KB_TRANSFER......         22.172
8500_B_TRANSFER             24.137
32_KB_TRANSFER.....         61.206
100_KB_TRANSFER....        170.710
100_KB_PIPES               172.160
2_MB_PIPES.........       3314.400

RPC...TCP...no_security
NULL_TRANSFER......          3.869
1_KB_TRANSFER......          6.633
2_KB_TRANSFER......          8.321
4_KB_TRANSFER......         11.442
4350_B_TRANSFER......       12.754
8_KB_TRANSFER......         17.804
8650_B_TRANSFER......       19.236
32_KB_TRANSFER......        56.238
100_KB_TRANSFER....        162.880
100_KB_PIPES               165.020
2_MB_PIPES.........       3344.400

RPC...UDP...connect
0_KB_TRANSFER......          4.531
4_KB_TRANSFER......         12.500
8_KB_TRANSFER......         22.906

RPC...UDP...call
NULL_TRANSFER......          4.480
4_KB_TRANSFER......         12.480
8_KB_TRANSFER......         22.880

RPC...UDP...packet
NULL_TRANSFER......          4.498
4_KB_TRANSFER......         12.575
8_KB_TRANSFER......         22.857


RPC...UDP...pkt_integrity
```

```
NULL_TRANSFER......          4.815
4_KB_TRANSFER......         15.390
8_KB_TRANSFER......         27.186


RPC...UDP...pkt_privacy
NULL_TRANSFER......          5.412
4_KB_TRANSFER......         25.280
8_KB_TRANSFER......         38.459

RPC...UDP...LOCAL
NULL_TRANSFER......          2.776
1_KB_TRANSFER......          3.306
2_KB_TRANSFER......          3.529
4_KB_TRANSFER......          3.898
8_KB_TRANSFER......          6.493
32_KB_TRANSFER.....         18.251
100_KB_TRANSFER....         50.196

RPC...TCP...LOCAL
NULL_TRANSFER......          2.554
1_KB_TRANSFER......          3.144
2_KB_TRANSFER......          3.276
4_KB_TRANSFER......          3.525
8_KB_TRANSFER......          4.798
32_KB_TRANSFER.....         11.965
100_KB_TRANSFER....         33.573

CELL_DIRECTORY_SERVICE
LIST_DIR...........       1002.400
SHOW_DIR...........       1033.400
LIST_OBJECT........       1002.400
SHOW_OBJECT........       1030.100
LIST_DIR_DCECP.....       1409.200
SHOW_DIR_DCECP.....       1440.900
SHOW_OBJ_DCECP.....       1441.200

SECURITY...DCECP...
VIEW_ACCOUNT.......       1565.800
VIEW_PRINCIPAL.....       1597.100
INTERACTIVE_GET....       1973.300
LOGIN_(sec)........       1065.000

THREAD.....tests...
CONDITION_VARIABLE.          0.401
THREAD_CREATE/TERM.          2.468
100_LOCK/UNLOCK....          1.409
```

Appendix B: Acknowledgements

I would like to thank Dave Bachmann for his input on the contents of this paper. I would also like to thank Tom Jones for generating the actual measurements, and for his help in getting the paper into suitable format for the Web.

Appendix C: Biographies

Debra A. Berding is a Staff Programmer. Currently, she is working in IBM Austin, TX, as a performance analyst on AIX and OS/2 DCE. She has an MS degree in computer science from Marist College.


```
Phone Mail:  (512) 838-1790
Internal  :  T/L   838-1790
email     :  dberding@vnet.ibm.com
```


Cindy Corn is a Staff Programmer. She is currently working as a performance analyst on AIX and OS/2 DCE. She has a BS in computer science from the University of Illinois at Urbana.

Tim Li is a Senior Programmer. He holds a BS degree in electrical engineering from National Taiwan University and a PhD degree in electrical engineering from Purdue University at West Lafayette, Indiana.

Ray Pekowski is an Advisory Programmer responsible for DCE RPC and threads performance analysis. He holds a BA degree in computer science from the University of Texas at Austin.

Bob Santeford is an Advisory Programmer responsible for DCE CDS performance analysis. He holds a BS degree in mathematics from the University of Houston at Clear Lake City, Texas.