



Introduction to OSS ~ & Porting to OSS

Bill Woo
NonStop Integrity Servers Product Mgmt

© 2004 Hewlett-Packard Development Company, L.P.
The information contained herein is subject to change without notice



Objectives



- Discuss the purpose of the Open System Services (OSS) subsystem.
- Describe the differences between UNIX and OSS.
- Compare the Guardian interface to the OSS interface.
- List the products that require OSS.
- Porting

Characteristics of Open Systems



- Compatibility — Applications running on the system will be able to run on future versions of the system.
- Portability — Applications running on the operating system on a given hardware platform will run on any vendor's system that utilizes that same operating system.
- Scalability — Applications written for the system will run on the full range of computer architectures and sizes.
- Interoperability — Applications running on the system will be able to communicate with any other system using the same networking protocols.

3

An open system must provide these characteristics:

- Compatibility
- Portability
- Scalability
- Interoperability

What factors can impact these characteristics?

- Selection of a processor and the development and customization of the instruction set
- Selection and customization of the operating system
- Selection and manner of implementation of software and hardware communication facilities, standards, protocols, and profiles
- Choice of high-level languages, databases, and office automation packages
- Degree of availability of information to others
- Consistency of hardware and software customization
- Consistency of end-user shell and script customization
- Consistency of the application development methodology and toolset

Objectives of Open System Services (OSS)



Provide a UNIX-like environment with transparent access to NonStop Kernel fundamentals for applications based on:

- CORBA, TUXEDO, and Pathway transaction servers
- SQL and Enscribe database engines
- J2SE applications and J2EE application servers
- XML and SOAP technologies
- web-enabling client/server technologies

Enable customers to reap the benefits offered by open, standards-based environments:

- development and management skill set
- large software base
- portability

What Is OSS?




- NonStop Kernel Open System Services
- API, command interpreter, and utilities
- NonStop system implementation of POSIX open standards
- Familiar interface and capabilities on top of NonStop system fundamentals

5

The term OSS refers to a set of facilities and capabilities introduced into the HP NonStop server.

OSS consists of a command interpreter (the Korn shell), utilities, and an API set. The facilities extend the standard NonStop server features by providing an open product.

Standards and Program Portability



Mainly influenced by:

- ANSI/ISO
 - C language (also Ada, COBOL, and FORTRAN)
- IEEE/ANSI/ISO
 - APIs (POSIX.1)
 - Commands and utilities (POSIX.2)
- X/Open
 - X/Open Portability Guide (XPG)

6

Formal standards are set by national or international standards bodies, such as the American National Standards Institute (ANSI) and the International Standards Organization (ISO).

Standards are also set by industry bodies representing vendors and/or users. X/Open is such an organization.


The major standards that currently influence application program portability at the system level are:

- ANSI/ISO C
- IEEE/ISO POSIX
- X/Open XPG4

Portability allows for:

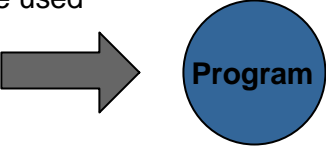
- Exchange of data from one system or application to another. Examples include data on diskettes, compressed files, and X.400 e-mail messages.
- The transfer of an application from one system to another. This means that the APIs (operating system, networking, database, and so on) must be supported on the target system. The target system must also support the run-time environment, including system libraries of common functions.
- Internationalization (I18N), the ability to customize an application to a different (human) language environment.
- The transfer of programmer skills (such as experience in C programming) and tools (editors, and so forth) to different vendor systems.

POSIX.1 APIs



POSIX.1 (API examples)

- abort — Generate an abnormal process abort
- alarm — Schedule alarm
- clock — Report processor time used
- cos — Cosine function
- execv — Execute a file
- exit — Terminate process
- free — Free memory
- pause — Suspend process execution
- stat — Get file status
- time — Get system time



7

POSIX.1 is the commonly used term for IEEE standard 1003.1, which is also an ANSI/ISO standard.

The POSIX.1 standard covers the basic operating system services interface and includes:

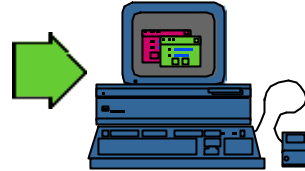
- Definitions and general requirements, such as limits and minimum values
- Process management functions and process environment
- Files and directories
- Input and output functions
- Data interchange formats—UNIX tar and cpio formats

POSIX.2 Commands



POSIX.2 (command examples)

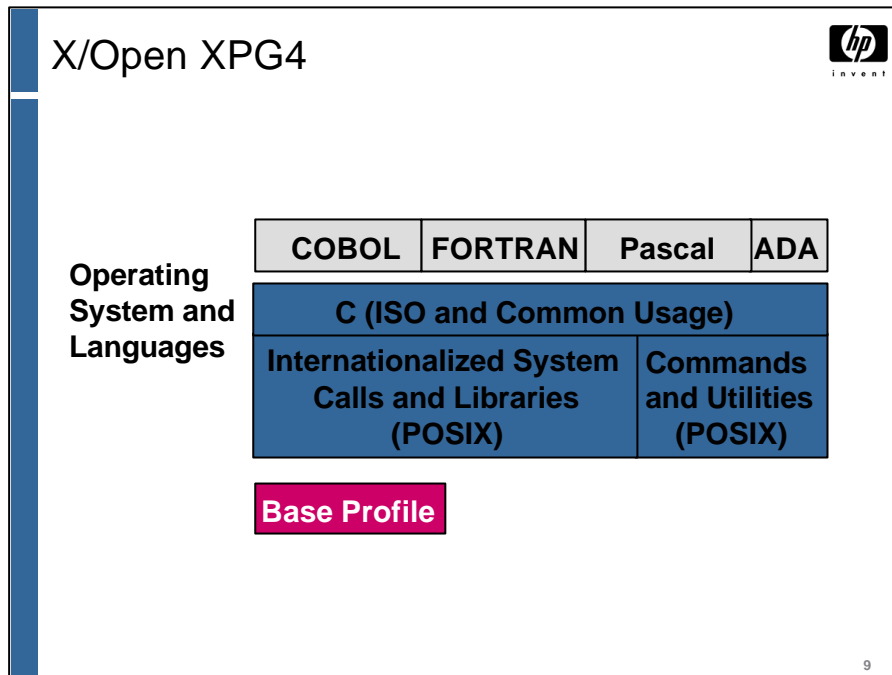
- cat — Concatenate and print files
- cp — Copy files
- date — Get the date and time
- grep — Search a file for a pattern
- ls — List directory contents
- man — Display manual pages
- rmdir — Remove directory



8

POSIX.2 is the commonly used term for IEEE standard 1003.2, which is also an ANSI/ISO standard.

POSIX.2 defines the shell (commands) and utilities. These commands are based on the UNIX SystemV shell with added functions from the Korn shell. More than 70 utilities are also defined.



X/Open, an industry association to promote open systems, publishes the X/Open Portability Guide (XPG), which contains recommendations of formal and de facto standards. The current version is the X/Open Portability Guide Version 4 (XPG4).

The base profile of XPG4 consists of:

- C compiler, based on the ISO definition or Common-usage C
- Systems calls and libraries, based on the POSIX.1 standard
- Commands and utilities, based on the POSIX.2 standard
- A model for internationalization (I18N), which refers to the process of developing programs for different languages, cultures, and code sets

SPEC 1170:

- Was a collaboration among major UNIX vendors, called COSE (Common Open Systems Environment), to create a common UNIX specification.
- Contained APIs selected from 50 of the most commonly used UNIX applications.
- Originally specified 1170 APIs (926 system calls, 70 header files, and 174 commands) based mainly on XPG4 and UNIX SVID3—the UNIX System V Interface Definition, 3rd Edition.
- Was passed from COSE to X/Open. Spec 1170 is now called the Single UNIX Specification and is published as the XPG4 Version 2 (XPG4.2) series.

What OSS Is Not



- UNIX
 - Similar but different
- POSIX
 - OSS is more than this
- A development environment
 - Facilities exist, but there are better ways

10

As well as knowing what OSS is, it is helpful to know what OSS is not:

- It is not UNIX—the UNIX kernel is not being used.
- It is not POSIX—it contains more than that.
- It is not intended as a development environment. There are, of course, facilities for compilation and debugging; but it is more efficient to do the editing and cross compilation on a workstation using TDS.

OSS Versus UNIX



- Which UNIX?
 - SVR4 versus BSD
- Whose UNIX?
 - HP/UX
 - TRU64
 - Solaris
 - AIX
- Like UNIX, but different
 - No UNIX kernel
 - No UNIX system administration
 - Only Korn shell

11

The OSS environment performs in many ways like a UNIX environment. However, there are two things to keep in mind:

- First, there are several versions of the UNIX operating system, all of which share some features but all of which differ in some ways.
- Second, although it might look like UNIX, OSS is not UNIX. There is no UNIX kernel and, most importantly, much of the system administration is performed using Guardian capabilities.

Products That Use OSS



- HP NonStop TUXEDO
- iTP WebServer
- Pathway/iTS
- HP NonStop SQL/MX
- HP NonStop CORBA
- All Java products
- HP NonStop SOAP
- Open Source Software ported to Open System Services (OSS):
 - Apache
 - Perl
 - TCL
 - and more ...

Myth: OSS is a Layer on Guardian

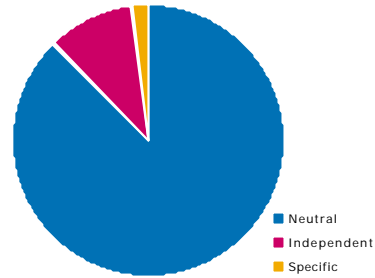


Reality: Much of the code is the same

- Same DP2, memory manager, interrupt handler, ServerNet services, TCP/IP, etc..

90% same code path for C/C++/COBOL

- environment-neutral:
sin(), strtod(), etc...
- environment-independent:
getc(), read(), etc...
- environment-specific (6):
fopen(), freopen(), remove(),
rename(), tmpfile(), tmpnam()



13

OSS Subsystem Components



OSS monitor: Provides SCF management interface, starts processes, mounts filesets

OSS name servers: Provide naming resolution services

TELSERV terminal server: Supports terminal sessions

Guardian SPOOLER: Provides printing facility in OSS (no native lp)

OSS pseudo-terminal utility server: Enables redirection of OSS process stdin, stdout, stderr to Guardian process

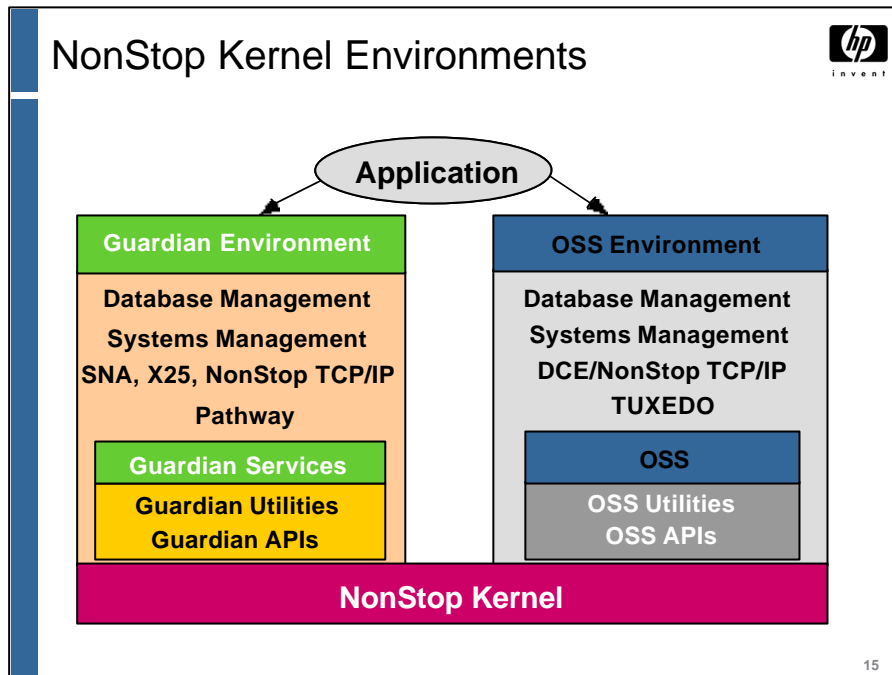
OSS file managers: Manages OSS CPU disk cache and pipe buffers

OSS pipe servers: Manages pipes created in each CPU

OSS message-queue server: Provides UNIX message queue support

OSS AF_UNIX local sockets server: Provides AF_UNIX socket transport

OSS TCP/IP transport agents: Provides conventional TCP/IP socket transport



The term “personality” was originally used to describe operating system APIs. People have applied the term to UNIX, DOS, Windows, Windows NT, and other user interfaces. However, NonStop systems provide for different environments meeting different needs at all levels of the system. For example, users need to have database, communications, messaging, and operating system environments.

NonStop Kernel Terminology



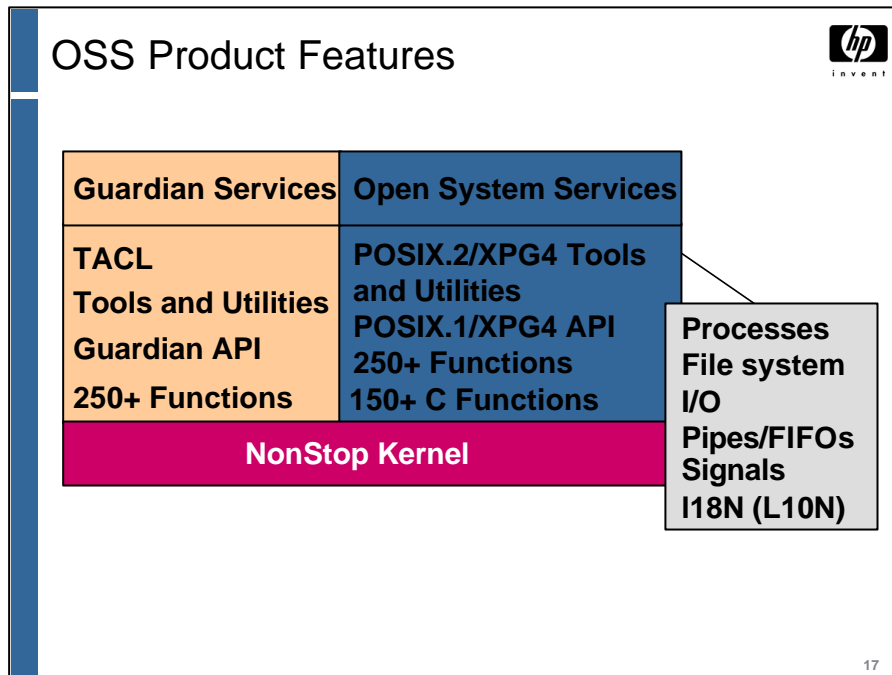
Commonly used terms

- Application program interface (API)
- Guardian
- Guardian environment
- Guardian personality
- Guardian services
- Open System Services (OSS)
- OSS environment
- OSS personality
- NonStop Kernel

16

The following terms are often used when describing the various components of the HP NonStop Kernel and its supported personalities:

- Application program interface (API) — The set of functions or procedures that permit access to services.
- Guardian — The original API to the NonStop Kernel.
- Guardian environment — The Guardian API, tools, and utilities (also referred to as “personality” in marketing literature).
- Guardian personality — The Guardian API, tools, and utilities (also referred to as “environment” in technical literature).
- Guardian services — An API to the NonStop Kernel and associated tools and utilities.
- Open System Services (OSS) — An API to the NonStop Kernel, and associated tools and utilities.
- OSS environment — The NonStop Kernel OSS API, tools, and utilities (also referred to as “personality” in the marketing literature).
- OSS personality — The NonStop Kernel OSS API, tools, and utilities (also referred to as “environment” in the technical literature).
- NonStop Kernel — The operating system for NonStop systems. The operating system does not include any application program interfaces.

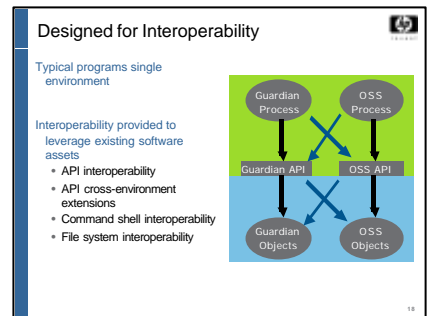


API = POSIX.1 + C library functions

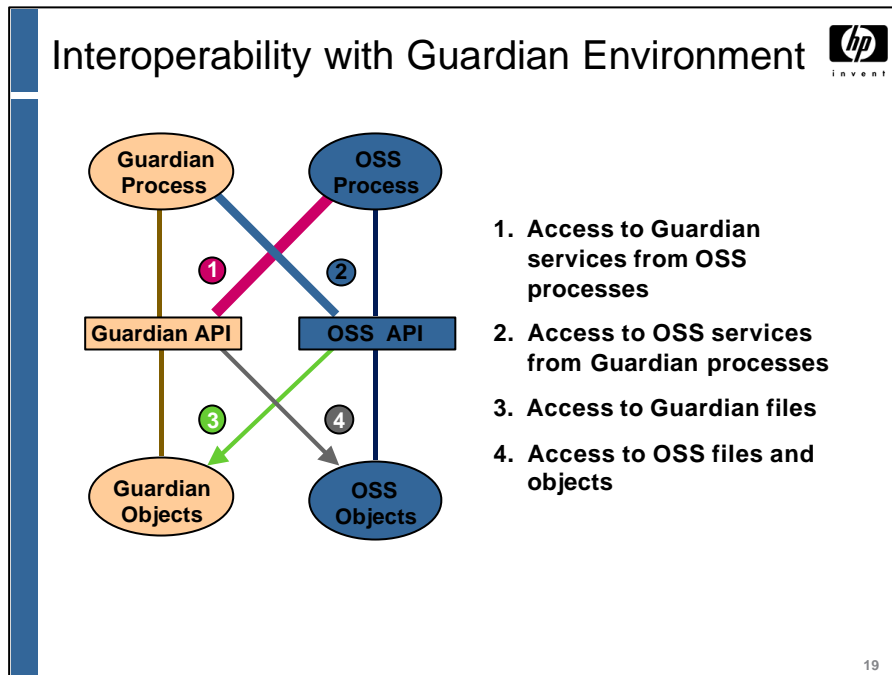
Tools = POSIX.2

The OSS environment incorporates:

- Different file system
- Different process model
- I/O mechanisms
- Pipes
- Signals
- I18N, providing national language support (NLS)



Typical programs either pure Guardian environment or pure OSS environment but extensive interoperability available for leveraging existing software assets



Because both Guardian and OSS environments now run over the NonStop Kernel, interoperability in this context refers to:

- The capability of a program to utilize the API of either environment
- The capability of an API to access and/or manipulate objects of the other environment
- The capability of utilities to access and/or manipulate objects of the other environment

In the diagram, the thickness of the lines indicates the relative amount of access provided. You can see that the most access to the Guardian API is from OSS processes, and the least access to Guardian objects is from the OSS API.

Interoperability provides a migration path for users.

OSS/Guardian Interoperability



- Terminals/Telnet session
- Common user ID scheme
- TACL/shell bridge
- Access to /G objects
- Access to /E objects
- Process status display
- Print job spooling
- File attribute display

20

There are many aspects of interoperability between the OSS and Guardian environments.

The OSS environment is normally accessed through Telnet. Terminals can include workstations, PCs, or terminals running VT-100 terminal emulation.

6530 terminals are normally used only for Guardian-based applications, including block-mode applications such as the TEDIT editor.

The user ID in the OSS environment must be associated with a Guardian user ID by means of the Safeguard subsystem.

Access to remote files on the Expand network is possible using /E.

Guardian and OSS “Better Together”



Interoperability with gtacl and grep

- gtacl allows OSS users to access the many utilities available in the Guardian environment
- Can be combined with the OSS text manipulation tools like grep to extract precisely the information needed, in any format

Examples


- I just started inetd to listen on port 514. How do I check it really does?

```
gtacl -c "scf;assume process \${zztcp}; status mon *" | grep 514  
TCP LISTEN 0.0.0.0 514 0.0.0.0 * 0 0
```

- How much memory my Java program is using?

```
gtacl -c "nskcom status swap-usage 3,352" | grep Heap  
Heap+Global+SRL+Stack 27MB 1787 27MB
```

OSS/Guardian Comparison



Guardian environment:	OSS environment:
<ul style="list-style-type: none"> • Process handle • Startup message • PARAMS, ASSIGNS, DEFINES • \$RECEIVE • <break> • System messages • Guardian I/O • Nowaited I/O • C run-time files, streams • Sequential I/O (SIO) • MOM, GMOM • progid • TAL, C, C++, COBOL85, PASCAL • Three-level file hierarchy • TACL • Edit and VS, TEDIT 	<ul style="list-style-type: none"> • Process ID • Inherited open files • args, env • Pipes • Job control signals • Signals • C run-time files, streams, shared files • groups, sessions, parent/child • setuid/setgid • C/C++, COBOL • Hierarchical path names • Shell • Vi editor

22

There are a number of differences between the OSS and Guardian environments. An OSS process has a number of attributes that are different from those of a Guardian process. For example, all OSS processes have a process ID, but Guardian processes are normally identified by their process handle. A new Guardian process normally inherits its parameters from the startup message, whereas an OSS process inherits the open files of its parent. The relationship between parent and child processes is indicated differently for Guardian processes than it is for OSS processes.

The normal text files in the Guardian environment are EDIT files, whereas the normal text files in the OSS environment are unstructured ASCII files. File-naming conventions in the Guardian environment are also more restricted than they are in the OSS environment. File names in the Guardian environment are not case sensitive, and they are restricted to eight characters. Names of OSS files are case sensitive.

The command interpreter in the Guardian environment is TACL, whereas the command interpreter in the OSS environment is the shell. TACL commands are not case sensitive, but OSS commands are.

Some level of interoperability has been provided between the two environments, such as:

- Invoking a command from the other environment
- Using an API from the other environment
- Operating on an object (file, process, and so forth) located in the other environment
- Communicating with another process in the other environment

OSS Environment

- Differences for Guardian users:
 - File system
 - Commands and utilities
 - Access through NonStop TCP/IP, or FTP, or NFS
 - Aliasing of spooler locations
- Differences for UNIX users:
 - System administration
 - Not “vanilla” (plain) UNIX
 - Use of OSS and Guardian sockets

23

For Guardian users:

- Hierarchical file system; flexible, case-sensitive file names
- Richer set of commands and utilities
- New editors—vi
- Only interactive access to the OSS environment is via a workstation or terminal using Telnet protocol
- Aliasing of printer names, instead of Guardian spooler names, such as \$\$.#xxx

For UNIX users:

- System administration performed through Guardian utilities—Safeguard and SCF
- Not all UNIX utilities are available; for example, du and df are not available
- Not all UNIX files are present; for example, /etc/passw and /etc/fstab
- Only certain devices are present in /dev; for example, /dev/tty and /dev/null
- Guardian sockets are available for backward compatibility

OSS sockets are compatible with Berkeley sockets.

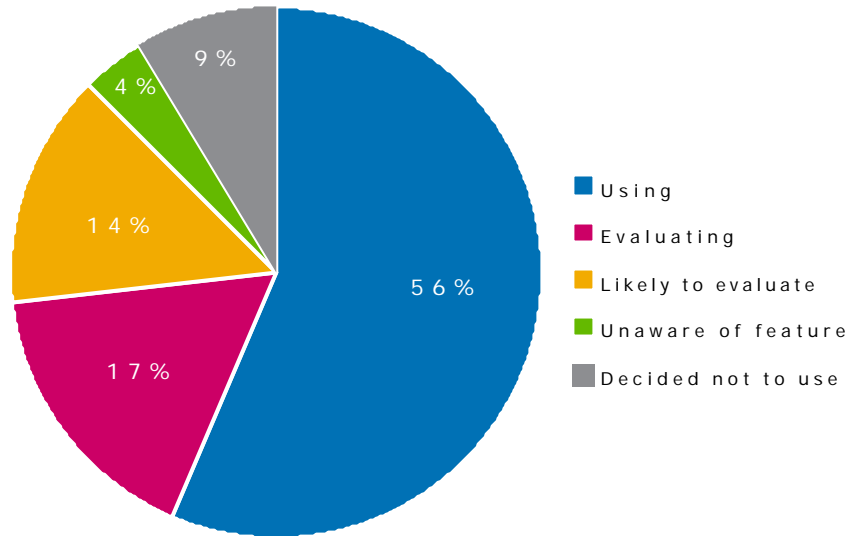
OSS Use Is Increasing Rapidly



- Half of the top 50 customers have OSS applications in production
- Most new applications are written for OSS
- Many partners are adding OSS support
- Why are customers using OSS?
 - Web-enabled applications
 - Provide open interfaces to applications
 - Add functionality to existing applications
 - Reduce development costs
 - Provide application portability
 - Developer availability



OSS Usage



Source: HP / ITUG Advocacy Working Group Online Survey, March 2003

Myth: OSS is Less Available than Guardian



Reality: OSS subsystem processes have the same availability as Guardian processes

Process pairs

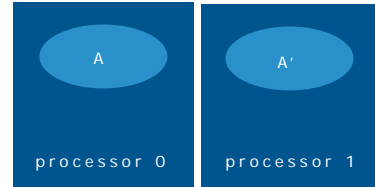
- OSS name servers
- OSS message-queue server
- OSS AF_UNIX local sockets server
- OSS pseudo-terminal utility server
- TELSERV terminal server

Persistent processes

- OSS monitor
- OSS TCP/IP transport agents

One-per CPU processes

- OSS file managers
- OSS pipe servers



OSS – New Availability Enhancements



OSS Persistent Processes (G06.24)

- Capability to place OSS processes under the control of the NonStop Kernel persistence manager
- Improves availability of standing processes (inetd, rexecd, user-written, etc...) by automatically restarting after a failure

OSS Online Fileset Configuration (G06.24)

- Enables online management of OSS fileset components and disks

Standard POSIX Threads (G06.24)

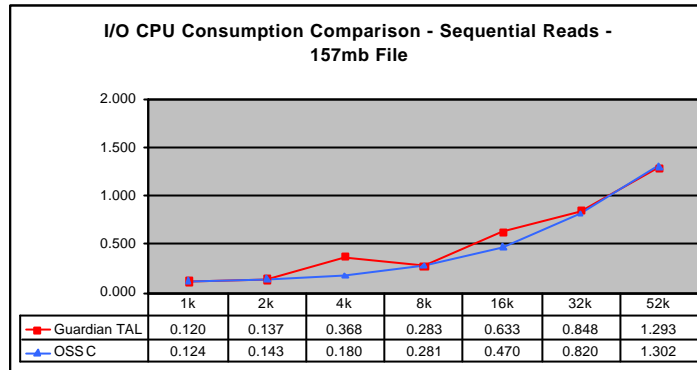
- Performance, context sensitive Pathsend support

27

Myth: OSS Performance is Much Worse Than Guardian Performance



Reality: Some OSS operations faster, some OSS operations slower; not an apples to apples comparison



28

Delivered Performance Improvements



OSS process creation [G06.12]
FTP server [G06.12]
FTP client [G06.14]
AF_UNIX sockets [G06.14]
Parallel library TCP/IP [G06.14]
NFS server (Parallel Library TCP/IP) [G06.14]
iTP Webserver v5.1 (Parallel Library TCP/IP) [G06.14/IP]
POSIX 1003.1c-compliant pthreads [G06.14, G06.18]
C/C++ heap manager [G06.15]
C/C++ compiler optimizer [G06.15]
NonStop TCP/IPv6 [G06.20]

29

Raising Limits to Handle Ever Growing Applications



OSS inode limit increased from 250,000 per fileset to 500,000 per fileset [G06.18]

OSS process limit increased from 16000 per system to 29000 per system [G06.19]

Guardian and OSS process limit increased from 2200 per CPU to 4000 per CPU [G06.19]

OSS open limit increased from 8000 per CPU to 12000 per CPU [G06.21]

OSS pipe/fifo limit increased from 256 per CPU to 1024 per CPU [G06.21]

More files, more processes, more I/O

DCT Limits extension (G06.23)



Destination control table entries increased from 32K to approximately 64K

May impact applications

Unsupported tool, DCTTOOL, available for testing of your application

Read support note S03123A

Myth: OSS is Hard to Install



Reality: OSS Easy Setup makes it a snap!

Enables a user with limited UNIX or Guardian knowledge to quickly install a working OSS subsystem

Scripts for installing and configuring a basic OSS environment

Non-interactive/interactive modes


- `TACL> RUN OSSSETUP DEFAULTS`

Many configuration parameters

Released in G06.15; runs back to G06.12 with IPMs




32

Myth: OSS is Hard to Manage 

Reality: Many new enhancements providing common Guardian/OSS management and UNIX features

- OSS Automatic Startup Service [G06.17]
- OSS Measure Support [G06.12, G06.17]
- DSM/SCM OSS Support [G06.18]
- OSS Pseudo Terminal Utility (OSSTTY) [G06.18]
- OSS File System Enhancements [G06.18]
- rexecd Remote Execution Server [G06.18]
- Performance management tools OSS Support (TPDC, Data Browser, Insight, DiskPro SPA [Summer 2003])
- inetd Network Services Server Load Balancing [G06.21]
- Third-party tools
- Open source tools

OSS – New Manageability Enhancements 

First release of new backup/restore architecture

Backup/Restore for OSS (G06.24)

- First release of new backup/restore architecture
- Common interface for Guardian, Enscribe, SQL/MP, SQL/MX, and OSS
 - Guardian, Enscribe and SQL/MP engine
 - SQL/MX and OSS engine

Removes need to use pax for OSS file backup/restore operations

Removes need to have multiple pax scripts to support large backup of large OSS filesets

34

OSS Automatic Startup Service




Automatic restart of OSS subsystem processes from system load, process failure, and processor failure and reload
Highly configurable from SCF; default is off

Improves manageability and availability

- Eliminates startup and shutdown scripts
- Eliminates operator intervention required after a system reload or failure
- Makes OSS subsystem available early in the system load process



35

OSS Measure Support 

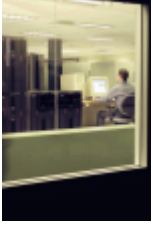
Built on OSS filename support released in G06.12

OSS subsystem instrumentation

- OSSCPU entity
- OSSNS entity

OSS file I/O support

- FILE entity supports OSS regular files, pipes, fifo sockets
- New OSS counters (read-bytes, write-bytes, more...)



34

DSM/SCM OSS Support



DSM/SCM support for OSS

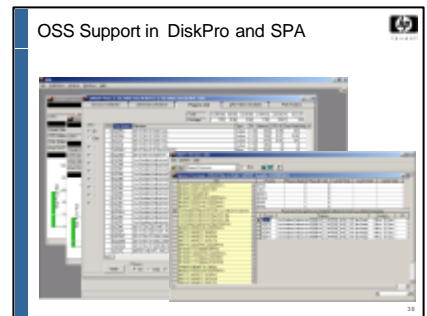
- Same installation and management of OSS files as Guardian files
- Optional (specify DSM/SCM option to activate)

Phased-in DSM/SCM support for OSS

- All SUT-based products (except NonStop DCE)
- Some independent products as new IP product versions are released
- IP products supported: ITP, WebServer, CORBA, SOAP, XML, and JMS

Benefits

- Provides single, consistent interface for installation of Guardian and OSS products
- Eliminates need to run COPYOSS/PINSTALL for software installation
- Enables DSM/SCM file audit capabilities for OSS files comparable to Guardian files
- Enables automated fallback
- Reduces outage window time for software installation



Both Guardian and OSS information is picked up, cross referenced, and displayed as needed by different functions

NonStop Delivers MORE SOFTWARE Manageability Enhancements coming



DSM/SCM (~G06.25)

Allow users to retain IPs and non-SUT software across coldloads

Backup/Restore 2

- SQL/MX tables (~G06.25)

Myth: OSS requires Safeguard



Safeguard is not required

Safeguard required if you need:

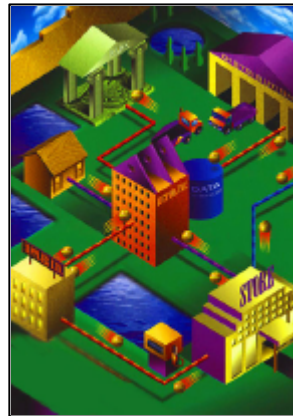
- OSS file security auditing
- FTP anonymous access and default to OSS file system
- UNIX style user names
- UNIX file-sharing groups
- UNIX initial directory and initial program set
- UNIX file permissions on the initial working directory

Depends really on business security requirements and how extensively OSS environment is used

OSS Applications in Production



- North American bank Internet banking system
- North American electronic communications network
- North American Telco billing system
- Asian stock exchange
- European railway reservation system
- European bank CRM system
- International ISP logon authentication



Advantages of Open System Services



“UNIX” with NonStop system fundamentals

- Fault tolerance
- Data integrity
- Parallel processing
- Scalable applications
- Networking

42

What does the OSS environment provide that is better than UNIX? OSS provides an open system and standards conformance, combined with the NonStop system fundamentals.

You get the familiar hierarchical file system with a fault-tolerant implementation using mirrored disks and controlled by a continuously available disk process.

Porting



- Discuss general porting considerations.
- Discuss porting design issues.
- Describe the different interprocess communication facilities.
- Describe some performance considerations.
- Write OSS programs that use \$RECEIVE.
- Configure Pathway serverpools to use OSS programs.

Porting Considerations (1 of 2)



- Scope of portability
- Levels of portability
 - Compiler compatibility
 - Data file compatibility
 - OS interface compatibility
- Availability of porting tools
 - lint
 - findcalls
 - CodeCheck
 - Open Systems Portability Checker (OSPC)
- Porting analysis
- Design alternatives for multiprocessing architecture
 - Using standard functions
 - Using equivalent functions
 - Using equivalent features

44

Before porting an application to the OSS environment, you must consider whether you intend to run the application on multiple UNIX environments, as well as on the OSS environment. If you do intend to run the applications on multiple UNIX environments, you must limit the use of NonStop system extensions in your application.

There are three levels of portability that must be considered when porting an application from a UNIX environment to the OSS environment: compiler compatibility, data file compatibility, and operating system interface compatibility.

The use of a proper set of tools can help you identify the areas that need the most attention when performing a port to the OSS environment. Commercial tools, such as Knowledge Software's Open Systems Portability Checker (OSPC) and Abraxas Software's CodeCheck, are suitable for porting applications to the OSS environment. The lint tool, common on UNIX machines, helps detect inconsistencies, and poor constructs. It is not available on NonStop systems; today the checking is done by compilers. The findcalls tool performs a simple lexical analysis of C source code by searching for the use of system calls. It identifies the system calls not supported by the target environment and therefore need to be replaced by equivalent function calls. It is available from your service provider.

Because NonStop systems use a loosely coupled multiprocessing architecture, different design alternatives might need to be considered when porting UNIX applications to the OSS environment.

You should use standard functions as much as possible to keep your application portable. If your application uses a nonstandard function, you might want to replace it with one of the standard functions. Some of the more commonly used nonstandard functions and their functional equivalents are listed in the OSS Porting Guide. In some cases, features commonly found in other UNIX environments are provided by equivalent functions in the OSS environment.

Porting Considerations (2 of 2)



- Porting to a uniprocessor
- Porting to a multiprocessor
- Using NonStop system extensions
- Using Guardian system procedures
- Using NonStop system features
- Redesign application
- Performance considerations
- Modular design
 - Design for portability

45

Most UNIX applications are written for a uniprocessor environment. When porting applications to the OSS environment, it is best to port the application as is first; then you can modify it to take advantage of a loosely coupled multiprocessor configuration at a later time.

If this program will not be ported back to a UNIX environment, you can use NonStop system extensions that are available to you from the OSS environment. This includes the use of the Guardian system procedures previously discussed. However, you are encouraged to place these extensions in separate modules so that they can be easily replaced for another UNIX environment.

In some cases, you might want to redesign the application to take better advantage of the rich functionality provided by HP NonStop systems. You must also be aware of performance considerations when using certain OSS and Guardian primitives.

Using Functional Equivalents



- ANSI/ISO C
- POSIX.1
- POSIX.2
- XPG3
- XPG4
- XPG4.2
- NonStop system extensions to OSS APIs
- New NonStop system OSS APIs
- Guardian APIs

46

The fewer nonstandard APIs used by a program, the more portable it will be for multiple UNIX environments, as well as to the OSS environment. An application becomes increasingly more difficult to port to multiple environments as it uses APIs in the categories in the order listed in this slide.

Applications that are restricted to the use of ANSI C run-time routines, and system interfaces and run-time routines defined in the XPG4 specifications, are generally very portable to most UNIX environments, including the OSS environment. The use of Guardian system procedure APIs will, of course, make the application non-portable to UNIX environments.

Using Equivalent Features



- Shared memory
- Semaphores
- Message queues
- Berkeley sockets, `select()`
- STREAMS, `poll()`
- Transport Level Interface (TLI/XTI)
- Memory allocation/layout
- Mapped files
- User-level threads

47

Some features that are commonly found in UNIX environments (but are not provided in the OSS programming environment, are implemented differently, or require some special considerations) are covered in this module. If your program uses these features, some changes might be required in the source code to use the equivalent features provided in OSS. The use of each of these equivalent features is discussed in the following slides of this module.

Interprocess Communication Using OSS and Guardian APIs



IPC method	Within a processor	Between processors	Between Expand nodes	With other systems
Guardian AF_INET sockets	Yes	Yes	Yes	Yes
OSS AF_INET sockets	Yes	Yes	Yes	Yes
OSS AF_UNIX sockets	Yes	Yes	No	No
Pipes (with OSS processes only)	Yes	Yes	No	No
FIFOs	Yes	Yes	Yes	No
Message queues (with OSS processes only)	Yes	Yes	No	No
OSS signals	Yes	Yes	No	No
Guardian shared memory	Yes	No	No	No
Guardian binary semaphores	Yes	No	No	No
OSS shared memory (with OSS processes only)	Yes	No	No	No
OSS semaphores (with OSS processes only)	Yes	No	No	No
\$RECEIVE	Yes	Yes	Yes	No

48

Most of the general IPC mechanisms used on UNIX platforms are supported in the OSS environment, at least on one processor. We will discuss each of these IPC mechanisms in more detail in subsequent slides in this module.

The Guardian version of sockets is available in the Guardian and OSS environments.

Shared Memory



- Shared memory in UNIX:
 - `shmget()`, `shmat()`, `shmctl()`, `shmdt()`
- Effective only on a single processor
- Functions provided in OSS
 - `shmget()`, create new segment
 - `shmat()`, attach segment
 - `shmdt()`, detach segment
 - `shmctl()`, remove segment
- Shared memory utilities
 - `ipcs`, obtain status of shared memory facility
 - `ipcrm`, remove shared memory identifiers

49

By default, one user data segment can be addressed by an OSS program. This segment is called the “heap” segment. The use of the UNIX shared memory primitives allows a program to add additional “flat” segments to its virtual address space. All shared memory segments are addressable at the same time.

A native mode OSS program can add any number of flat data segments to its address space—up to the maximum available address space. Each flat segment is allocated to a 128 KB boundary.

A nonnative mode program can add only up to thirteen (13) flat data segments to its address space. They are allocated to a 32-MB boundary.

This supports the implementation of SVR4 shared memory on a single processor. The use of these primitives is discussed in the Open System Services System Calls Reference Manual.

Semaphores



- Semaphores in UNIX:
 - semctl(), semget(), semop()
 - Used in conjunction with shared memory
- Effective only on a single processor
- OSS can use UNIX counting semaphores or Guardian binary semaphores
- Guardian semaphores
 - System-level function
 - Binary semaphore, not a counting semaphore
 - Use not recommended in the OSS environment

50

A semaphore is a data structure that is shared by several processes. It is often used to synchronize operations when multiple processes access a common, nonshareable resource.

Semaphores that control access to a single resource are called binary semaphores (takes on a value of 0, for resource in use, and 1 for resource is available).

Semaphores that control access to multiple resources are often called counting semaphores. They assume a range of non-negative values.

UNIX defines a set of counting semaphores that are commonly used in UNIX system implementations. These semaphores are most often used in conjunction with the use of shared memory. They are provided in the OSS environment as well.

Binary semaphores are supported using Guardian procedure calls. However, it is not recommended that these be used in conjunction with the use of the shared memory functions.

Message Queues



- Message queues are defined in UNIX
- Implemented in OSS
 - msgctl(), msgget(), msgsnd(), msgrcv()
 - Can be called from OSS program only
- Guardian messages available through the use of \$RECEIVE and WRITEREAD
- Alternative approaches:
 - Redesign application to use another IPC mechanism; for example, pipes, signals, FIFOs, and sockets
 - Use equivalent Guardian functionality; for example:
 - \$RECEIVE/WRITEREAD
 - System-level messages (not recommended)

51

XPG4 user-level message queues are supported in the OSS environment. Not many UNIX applications currently use message queues. They tend to use other IPC mechanisms that are generally available on all UNIX systems. Some of the alternatives are listed above in this slide.

NonStop systems also support message queues, at both the system level and the user level. OSS applications can use Guardian message queues, although the functionality provided is somewhat different from that of UNIX message queues.

OSS message queues are fault tolerant by default. Application programmers must specify if they want to use message queues that are not fault tolerant.

Memory Allocation



- Standards-compliant programs use `malloc()` and `free()`
 - Old programs using `brk()`, `sbrk()`, and `alloca()` must be recoded to use `malloc()` and `free()`
- Memory available for native OSS process
 - Expandable user stack segment = 32 MB maximum
 - Expandable user “heap” data segment
 - Large number of shared flat segments, 128-KB boundary
 - Heap + global data + stack + flat segments = max 1120 MB
- Memory available for nonnative OSS process
 - Maximum of 64 KB for user stack
 - Maximum of 127.5 MB for user “heap” data segment
 - Maximum of 13 shared flat segments, 32-MB boundary

52

Some existing programs might still use the `brk()`, `sbrk()`, and `alloca()` memory allocation routines. These routines must be replaced by the standards-compliant `malloc()` and `free()` routines.

Each native OSS program has a stack that is dynamically expandable up to a maximum of 32 MB. For additional space requirements, a process can access the heap or acquire, by using the Guardian API `SEGMENT_ALLOCATE_`, flat segments. The total data space available to the process for the stack, the heap, global data, and flat segments is 1120 MB.

For non-native OSS programs, the stack segment is limited to 64 KB. This might present a limitation for programs that use deep subroutine recursions. Programmers need to make sure that they allocate large structures in global memory rather than on the stack. Up to 13 segments can be attached at a time using the shared memory system calls. These restrictions do not apply to native OSS programs.

Mapped Files



- Establishes mapping between the process' address space and the object represented by file descriptor fd
- Implemented in SVR4 using `mmap()` and `munmap()`
- Are part of standard UNIX
- Not supported in OSS
 - Redesign application

53

Mapped files provide a mechanism for a process to access files by directly incorporating file data into the address space of the process. After a file is mapped into a process address space, the data can be manipulated as memory. If more than one process maps a file, its contents are shared among them. This feature is implemented in UNIX System V Release 4 (SVR4) but is not part of the XPG4 specifications.

In general, most UNIX programmers do not use this feature, but if your program does use mapped files, you will have to redesign it to use a less transparent file access mechanism, because mapped files are not supported in the OSS environment.

Threads



- Multiple threads of execution in a process' address space
- IEEE POSIX Standard 1003.1c pthreads implementation
- Available for native processes only
- Threads are non-preemptive
- Process resources are either global or private
- Compilation and Linking considerations
 - Header file and directory search path
- Jacket routines available for Pathsend calls and TMF usage
 - SPT_SERVERCLASS_SEND_()
 - SPT_SERVERCLASS_SEND_INFO_()
 - SPT_ABORTTRANSACTION()
 - SPT_BEGINTRANSACTION()
 - SPT_ENDTRANSACTION()
 - SPT_RESUMETRANSACTION()

54

Multithreading is a programming model that enables multiple threads of execution in a process's address space. Also, multithreading allows many sequential processing tasks to execute concurrently within a process (for example, a terminal control process). Multithreaded applications take advantage of a shared-memory, multiprocessor system.

Standard POSIX Threads is a user-space implementation of IEEE POSIX Standard 1003.1c pthreads. This is available to native C and C++ applications in the OSS environment on NonStop servers. Threads are scheduled for execution by the Standard POSIX Threads library, not by the NonStop Kernel. All threads created within a process share the same process address space.

With Standard POSIX Threads, one thread can never be preempted by another thread. Each thread executes until it relinquishes control either by explicitly calling the `sched_yield()` function or by waiting on a mutex or condition variable.

All resources of a process—for example, open files or memory—are either global or private. Private resources can be accessed only by a specific thread. Global resources can be accessed by all threads. Access to global resources by threads should be synchronized using mutex or condition variables.

Include the following header file: `/usr/include/spthread.h`

When compiling, your directory search path should include `/usr/include` and `/usr/include/spt`

You need to include `ZSPTSRL`, the Standard POSIX Threads shared run-time library, when linking with `nld`. This library is located in the current `sysnn` subvolume.

For more information on programming with Standard POSIX Threads, refer to the Open System Services Programmer's Guide.

Performance Considerations



- `fork()`, `exec()`
- `tdm_fork()`, `tdm_exec()`, `tdm_spawn()`
- `open()`, `close()`, `creat()`
- `read()`, `write()`
- `opendir()`, `readdir()`
- pipes across processors
- sockets/\$RECEIVE

55

When porting UNIX applications to the OSS environment or writing new applications for the OSS environment, you need to be aware of the performance implications of certain OSS operations. The system primitives listed here should be used carefully in a program. There might be cases where the application makes heavy use of the primitives listed above when the application should be redesigned to perform better.

Pipes and FIFOs



- Pipe in single processor
 - Parent and child processes execute in same processor
- Pipe across processors
 - Parent and child processes execute in different processors
 - Pipe management is done in parent's processor
- FIFOs
 - More likely that parent and child processes execute in different processors

56

When the parent process creates a pipe, the pipe management is done in the same processor that the parent process is running in. When the child process is created, the pipe file descriptors are inherited. If the child process is created in the same processor as the parent process, all pipe I/O between the two processes is handled within that processor. If the child process is created in another processor, all pipe I/O by the child process is handled by the processor in which the parent process is running. This requires interprocessor traffic and incurs more overhead than the case in which the parent and child processes are running in the same processor.

In the case of FIFOs, there does not need to be any direct parent-child relationship between processes that are communicating using a FIFO. Thus, there is a greater likelihood that the two communicating processes are running in different processors, unless care is taken to make sure that this does not happen.

The bottom line is that there is greater overhead associated with pipe and FIFO I/O operations when the communicating processes are running in different processors. The programmer needs to take this into consideration when designing or building an application.

Sockets and \$RECEIVE



- Sockets
 - Preferred interapplication communications mechanism for open systems
 - Both OSS and Guardian sockets are available
 - NonStop TCP/IP implemented as a process
- \$RECEIVE
 - Interapplication communications mechanism for Guardian processes
 - Implemented using NonStop Kernel message system
 - Commonly used by Guardian servers/requesters
 - Can be used between Guardian and OSS processes
 - Can be used between OSS processes

57

Both OSS and Guardian sockets are implemented using NonStop TCP/IP system processes in the NonStop Kernel. There is more overhead associated with using a process model than with using kernel device drivers as implemented in most UNIX systems. UNIX socket applications can receive SIGIO and SIGURG signals.

- SIGURG — Signal notifies process of pending urgent data.
- SIGIO — Signal notifies process when a socket has data to be read.

Guardian sockets do not support this mode of notification. Thus, different mechanisms must be used in OSS programs to achieve the same results. An application designer must keep these differences in implementation in mind when considering the performance impact of the design of the application.

The use of \$RECEIVE is an efficient interprocess communications mechanism when used between Guardian processes. This mechanism can also be used between Guardian and OSS processes and between two OSS processes, but the Guardian FILE_OPEN_() system procedure call must be used in the OSS program to do this, and the OSS processes must be “named.” The performance considerations when \$RECEIVE is used between two OSS processes should be no different than when \$RECEIVE is used between two Guardian processes.

Servers and Daemons



- inetd in the UNIX environment
- LISTNER in the Guardian environment
- Static servers
- Persistent servers

58

The inetd daemon is used on UNIX systems to start up and control servers that handle requests on specific port numbers. When a request comes into the inetd daemon, the configuration file is checked to determine which server to fork and exec to handle requests on that port number.

The equivalent functionality in the Guardian environment is provided by the LISTNER process. Guardian server processes are created and started with the appropriate parameters when an incoming request is received by the LISTNER process.

OSS servers can be started from either the Guardian or the OSS environment.

OSS servers can also be started using the inetd daemon in the OSS environment.

Static servers in the OSS environment work well when the connection request arrival rate is low, or the time required to provide the server is relatively long; for example, FTP and Telnet.

When the connection request arrival rate is high or service times are very short, the use of persistent servers is more appropriate. A static server can be used to pass requests to a set of persistent processes (created when the static server is first started up), thereby eliminating the fork()/exec() overhead associated with process creation.

\$RECEIVE Handling



- Same usage as within Guardian requesters/servers
- For servers:
 - Use FILE_OPEN_ to open \$RECEIVE
 - Use REPLY[x] to return response
- For requesters:
 - Use FILE_OPEN_ to open server process by name
 - Use WRITEREAD[x] for two-way communication
- Usage:
 - Server: \$ run -name=/G/<srv-name> <srvobj>
 - Requester: \$ run <reqobj>

Issue: How can requesters and servers be folded into the Pathway environment?

Pathway Specifics (1 of 2)



- Requester:
 - Consider using PATHSEND APIs
 - Replace FILE_OPEN_ and WRITEREAD[x] with:
 - SERVERCLASS_SEND_
 - SERVERCLASS_SEND_INFO_
- Server:
 - Can still use REPLY[x]
 - Needs PATHWAY configuration update

Pathway Specifics (2 of 2)



- Pathmon configuration:


```
set pathway maxlinkmons <n>
```
- Server configuration needs:


```
set server processtype OSS
set server cwd <OSS-pathname>
set server stdin <fname>
set server stdout <fname>
set server stderr <fname>
set server program <object-pathname>
```
- These conflict with processtype OSS:


```
set server IN <fname>
set server OUT <fname>
```

61

The CWD server configuration attribute is used to specify the absolute OSS pathname of the current working directory of an OSS server process. This value is used to resolve relative pathnames specified for other OSS server attributes in the server class. Case is significant. If you omit this attribute in the SET command, relative OSS pathnames are resolved using the default directory specified by the pathcom CMDCWD command. If no value is set for this attribute by means of either command, there is no default.

If the CWD attribute is not specified (in the CMDCWD command or the SET SERVER CWD command), the OSS stdin, stdout, and stderr files must be absolute OSS pathnames. Case is significant. If the file or files specified do not exist, it is created. If you omit this attribute, a null string is sent to the server process.

The IN and OUT server configuration attributes are meant for processtype Guardian only.

Getting the Most Performance in OSS



Configure multiple name servers

- Must access files using current working directory, not absolute pathnames
- Enables flexible control over buffering, caching, and auditing

Use Parallel Library TCP/IP or NonStop TCP/IPv6

- AF_INET sockets much faster
- Larger performance gain for OSS than Guardian applications

Use OSS sockets instead of Guardian sockets

Use new heap manager

Use standing processes instead of transient processes

Use persistent file opens instead of repeatedly opening and closing files

Getting the Most Availability in OSS



Automate tasks

- Use DSM/SCM instead of COPYOSS/PINSTALL
- Configure OSS Automatic Startup Service

Configure file systems correctly

- Multiple files sets for flexible control over buffering and caching


Use middleware for process management

- Pathway, TUXEDO, CORBA, J2EE

Use Guardian resources when it makes sense

- \$RECEIVE, DP2 Queue files, SQL, Enscribe

Avoid using shared memory

Open Source Software for NonStop 


Available from user group ITUG
www.itug.org in ITUGLIB

Goal is to encourage community support / enhancement of open source

Currently about 170 packages ported:

- Editors: vim, nano, emacs, ed
- Languages: perl, python, tcl
- File sharing: samba
- Web server: apache
- Security: openssh, openssh, sudo
- Productivity tools: GNU make, cscope

Source, executable, and documentation provided



FLOSS released as open source as library and tools to make porting easier

Potentially most Open Source can be ported easily since OSS is a Posix platform with a "unix" personality and a set of macros to allow using c89 compiler transparently when re-compiling software

64

Open Source Manageability Tools



command shells: bash, gt, tcsh,
zsh osh (operators shell)
version control systems: cvs, rcs,
scs
utility sets: diffutils, fileutils,
findutils, gt, mtools, sh-utils,
sharutils, textutils
utilities: less, rsync, uucp, which
development tools: cscope,
cppunit, dmalloc

editors: ed, emacs, vim
file server: samba
printing tools: a2ps, ifhp,
LPRngp, LPRngTool
security tools: openssh, openssl
scripting languages: perl, python,
tcl
file compression tools: bzip2,
gzip, zlib
windowing systems: X11, vnc

Open Source Development Tools



parsing tools: flex, bison

database routines: gdbm

mathematic routines:

- GLPK -- GNU linear programming library
- GMP -- GNU multiple precision arithmetic library
- GSL -- GNU scientific library

cryptographic routines: libgcrypt, openssl

character encoding routines: libiconv

XML/webservices support routines: libxml2, libxslt, gsoap

UNIX Curses routines: ncurses

data compression routines: zlib

Open Source @ ITUGLIB (Sept 04)				
A2ps	DHCP	Glib	Less	Mktemp
Amanda	Dicton	Global	Lfp	Mobi
Apache	Diffutils	GLPK	Libcrypt	Miscutils
Apache-ant	Emulab	Gmp	Libiconv	Nano
Aspell-en	Doschk	Gnub	LibIDL	Nano
Atk	Eel	Gnuage	Liblmg	Ncftp
Autocore	Emacs	GnuPG	Libngp	Norfses
Automake	Enscript	Gperf	Libpcap	Nek-SNMP
Barcode	Expat	Grp	Libpng	Nep
Bash	Expect	Gcc	Libsigsegv	Nfs
Bc	File	Gccap	Libtool	Oleo
Bind	Fileutils	Gid	Libvorbis	OpenLDAP
Bison	Findutils	Gypgit	Libwm	OpenSSH
Bool	Flex	Gulp	Libxml2	OpenSSL
Bzip2	Flex	Gulp	Libxml2	OpenSSH
Cac	Fontconfig	Helio	Libxslt	Panda
Copunit	Freetype	Help2man	LPRng	Pango
Cscope	Gawk	HTTpd	Lynx	Patch
CSSD	Goal	Httpd	M4	Perl
Curl	Gdm	Iftp	Make	Perl
CVS	Gengenopt	Index	Man-db	Popt
DAP	Gettext	Jdk	Man-db	Popt
Db	Ghostscript	Jikes	Marst	Popt
Domanual	Git	Jpeg	MC	Python
			Miscfiles	Python
				RCS
				Readline
				Recode
				TeX
				Time
				Tk
				Tracetest
				Udp
				Sane-backends
				UMLinux
				Umap
				Vim
				Vnc
				Wdiff
				Wget
				Which
				Wikipedia
				Xaos
				X11
				Stow
				SNACC
				Sudo
				Stunnel
				TeX
				TeX
				Xrander
				Zsh
				Zope
				Termcap
				Termutils
				Zsh

Open Source Tools – vim



Editor like vi except that it:

- Resizes your window automatically
- Assigns different colors to different objects in your C, C++, Java, and user-defined source code (comments in blue, strings in red, etc...)

```
C:\WINNT\system32\cmd.exe
#define MAX_CPUS 15
/*
 * exec? - ToDo
 * execl? - ToDo
 * execlp - ToDo
 */
extern char **environ;
int
loss_execv(const char *path, char * const argv[])
{
    int rv;
    int number_of_cpus = MAX_CPUS;
    struct process_extension pe_parms = DEFAULT_PROCESS_EXTENSION;
    struct process_extension_results pr_results =
        DEFAULT_PROCESS_EXTENSION_RESULTS;
```

68

Open Source Tools – cscope



Browses C and C++ source code files for specific elements of code

```

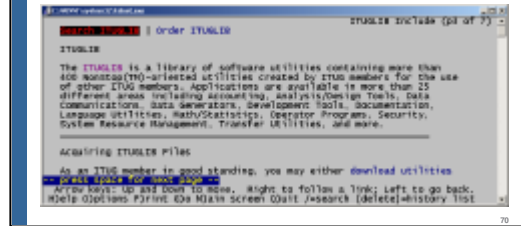
cscope: read
cscope: file.c
file      function
file.c    flock_read  200 flock_read(file, void *buffer,
file.c    flock_read  204 bytes * flock_read(file, void *buf),
file.c    read        0 flock_read(file, buffer, (bytes))
* 2 more lines - press the space bar to display more *
Find this C symbol:
Find this global definition:
Find functions called by this function:
Find functions calling this function:
Find this text string:
Search this text string:
Find this symbol pattern:
Find this file:
Find files including this file:

```

Terminology.

Open Source Tools – lynx

Text mode web browser

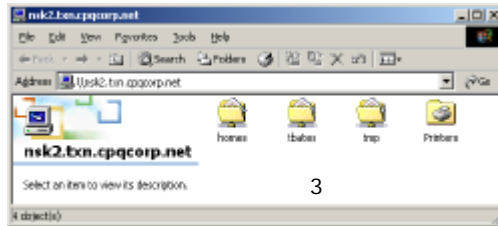
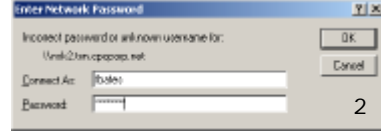
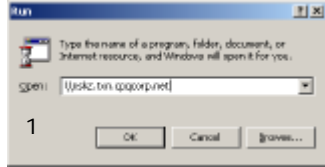


Terminology.

Open Source Tools – samba



File server for SMB/CIF clients (like Windows)



71

Open Source Tools to Make OSS Easier for Guardian Users 

- EDIT**
 - Starts Guardian EDIT from an OSS shell to edit an OSS file
- FC**
 - Provides Guardian FC (fix command) in the OSS shell
- GHSTACL**
 - Runs GTACL command from the OSS shell without requiring additional user authentication
- TEDIT**
 - Starts Guardian TEDIT from an OSS shell to edit an OSS file

Available at www.greenhouse.de



72

- **EDIT - FC - GHSTACL - TEDIT** into a subvol,

Open Applications in Production



- **Banking / Finance**
 - financial trading (Europe)
 - Internet banking (NA)
 - integration engine (NA)
- **Telecommunications**
 - wireless 911 (NA)
 - wireline telco billing (NA)
 - network switch provisioning (Worldwide)
 - prepaid billing (Europe)
- **Stock exchanges** (Asia, Europe)
- **Travel**
 - car rental operator reservation system (Worldwide)
 - airline application (NA)
 - reservation system (Worldwide)
- **Retail**
 - CRM and fraud prevention (NA)



73

Learning More

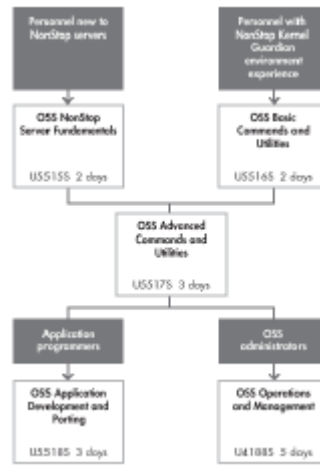


Manuals

- OSS Programmer's Guide
- OSS User's Guide
- OSS Management and Operations Guide

Technical Update
Training CDs

New classroom-based
training curriculum path



Our Commitment to OSS



OSS is now the primary NonStop application infrastructure and OS environment for all NonStop middleware and database development.

Improve NonStop fundamentals in OSS.

- We're delivering incremental improvements.

Make OSS comply with POSIX/LINUX standards.

Two porting centers – Cupertino and EMEA.

We plan a major, long-term investment in OSS.

- NonStop 64-bit OS will support only OSS applications.

OSS Roadmap Scope



Complete NonStop Open-System Operating Environment

- OSS process control, memory management, & signals
 - OSS System Limits
 - Threading Environment
 - mmap
- OSS file system and associated disk-process functions
 - Transparent DP2 takeover for OSS files
 - OSS files > 2GB
 - ACLs for OSS files
- OSS networking and communications
 - Non-blocking Term I/O
- OSS security infrastructure
 - UNIX 98 Std Security APIs & ACL Infrastructure
- OSS tools and utilities
 - On-going porting of Utilities & Tools
- Support for OSS-based middleware products

76

