Cisco IOS XR Fundamentals

ciscopress.com

Mobeen Tahir • Mark Ghattas
Dawit Birhanu • Syed Natif Nawaz

Cisco IOS XR Fundamentals

Mobeen Tahir, CCIE No. 12643
Mark Ghattas, CCIE No. 19706
Dawit Birhanu, CCIE No. 5602
Syed Natif Nawaz, CCIE No. 8825

Cisco Press

800 East 96th Street

Indianapolis, IN 46240

Cisco IOS XR Fundamentals

Mobeen Tahir, Mark Ghattas, Dawit Birhanu, Syed Natif Nawaz

Published by:
Cisco Press
800 East 96th Street
Indianapolis, IN 46240 USA

Printed in the United States of America

First Printing June 2009

Library of Congress Cataloging-in-Publication Data:

Cisco IOS XR fundamentals / Mobeen Tahir ... [et al.].

p. cm.

Includes bibliographical references.

ISBN-13: 978-1-58705-271-2 (pbk.)

ISBN-10: 1-58705-271-7 (pbk.)

1. Cisco IOS. 2. Routing (Computer network management) 3. Routers (Computer networks)
4. Internetworking (Telecommunication) I. Tahir, Mobeen, 1966- II. Cisco Systems, Inc. III. Title.

TK5105.8.C57C548 2009

004.6—dc22

2009019283

ISBN-13: 978-1-58705-271-2

ISBN-10: 1-58705-271-7

Warning and Disclaimer

Trademark Acknowledgments

Corporate and Government Sales

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact: **U.S. Corporate and Government Sales** 1-800-382-3419 corpsales@pearsontechgroup.com

For sales outside the United States please contact: **International Sales** international@pearsoned.com

Feedback Information

At Cisco Press, our goal is to create in-depth technical books of the highest quality and value. Each book is crafted with care and precision, undergoing rigorous development that involves the unique expertise of members from the professional technical community.

Readers' feedback is a natural continuation of this process. If you have any comments regarding how we could improve the quality of this book, or otherwise alter it to better suit your needs, you can contact us through email at feedback@ciscopress.com. Please make sure to include the book title and ISBN in your message.

We greatly appreciate your assistance.

Publisher: Paul Boger

Associate Publisher: Dave Dusthimer

Executive Editor: Brett Bartow

Managing Editor: Patrick Kanouse

Development Editor: Dayna Isley

Project Editor: Tonya Simpson

Editorial Assistant: Vanessa Evans

Book Designer: Louisa Adair

Composition: Mark Shirar

Indexer: Ken Johnson

Business Operation Manager, Cisco Press: Anand Sundaram

Manager Global Certification: Erik Ullanderson

Copy Editor: Mike Henry

Technical Editors: Mukhtiar Shaikh, Syed Kamran Raza

Proofreader: Leslie Joseph

About the Authors

Mobeen Tahir, CCIE No. 12643 (SP, R&S), is a network consulting engineer with the World Wide Service Provider Practice team in Cisco. Mobeen started his career in the communication industry in 1993 with France-based Alcatel. While working for Alcatel between 1993 and 1998, Mobeen engaged in assignments ranging from manufacturing voice switches to planning large-scale telecommunications projects. He joined Cisco in 1999 and has worked on the development testing of the IOS XR operating system for c12000 and CRS-1 platforms. His current role as a network consulting engineer at Cisco consists of designing and deploying NGN networks in the service provider space. Mobeen specializes in IOS XR–based deployments and provides consulting services to Cisco customers. Mobeen has attained master of engineering and B.S.E.E degrees from institutions in Canada and the United States. He lives with his wife and two children in Cary, North Carolina.

Mark Ghattas, CCIE No. 19706 (Service Provider), is a solutions architect focusing on architecture and design. He manages the World Wide Service Provider NGN Core Practice team in Advanced Services. Mark has more than 15 years of experience with data communication technologies. Mark joined Cisco Systems in 1999 and has supported strategic service providers. Mark has supported many of the first CRS-1 customers in Japan and the Asia Pacific theatre, CANSAC, Latin America, and North America. He has presented on various topics at Networkers relating to IOS XR. He holds a bachelor's degree from the University of Maryland and plans to earn his MBA degree.

Dawit Birhanu, CCIE No. 5602, is a technical leader with the World Wide Service Provider Practice team in Cisco Systems, where he is responsible for assisting global service providers with the deployment of new NGN products and technologies. He specializes in IOS XR–based platforms, QoS, MPLS, and BGP. Dawit joined Cisco Systems in 2000 and has worked on the deployment of new technologies for Cisco 12000 and CRS-1 in the service provider space. Dawit has a master of telecommunications degree from the University of Pittsburgh and a master of electronics engineering degree from Eindhoven University of Technology, The Netherlands. Before getting into the networking industry, Dawit was a lecturer of electrical engineering at Addis Ababa University, Ethiopia, between 1992 and 1995. Dawit lives with his wife and two daughters in Raleigh, North Carolina.

Syed Natif Nawaz, CCIE No. 8825 (SP, R&S), has more than ten years of experience in providing networking design, deployments, and escalation assistance to various service provider customers. Syed Natif Nawaz is currently the IOS XE software development manager at Cisco Systems, where he works on customer-focused software qualification/certification/deployment, feature integration, release processes, and other software quality initiatives. He has presented on various MPLS-related topics in the Networkers conference (Florida), MPLS Power Sessions (London), NANOG (Dallas), and APRICOT (Perth) and has contributed to articles such as "L2VPN: Changing and Consolidating Networks" in *Techworld* and "Cell Packing" in *Packet Magazine*. Formerly, Syed Natif Nawaz worked as a development engineer at Assured Access technologies and Alcatel, where he developed software for access concentrators. In addition to higher education in electrical and electronics from the University of Madras, Syed Natif Nawaz also holds an M.S. in computer science and engineering from State University of New York at Buffalo.

About the Technical Reviewers

Mukhtiar Shaikh is a distinguished services engineer at Cisco and a senior member of the central engineering team within the Customer Advocacy Organization. He joined Cisco in October 1996. During his early years at Cisco, he provided technical support to Cisco's large ISP accounts. His areas of focus are IP routing protocols, multicast, and MPLS technologies. Over the past several years, he has led various design projects and has been involved in the deployment of MPLS in the service provider and Enterprise NGN networks. In his current role, he provides technology leadership and architectural and design consulting to the Cisco Advanced Services accounts. Mukhtiar is a regular speaker at various industry forums. He is a CCIE and holds an M.S. degree in electrical engineering from Colorado State University.

Syed Kamran Raza is a technical leader (MPLS software) at Cisco Systems. He joined Cisco in 2000 to work on MPLS architecture and design for Cisco IOS XR and the carrier grade core router platform (CRS-1). For the past eight years, he has been priming the IOS XR MPLS LDP software development and has contributed to various features, including RSVP, LDP, MPLS forwarding, MPLS-based L2/L3 VPNs, SRP, and High Availability. Prior to Cisco, he worked as a software designer at Nortel Networks and as a telecommunications engineer at Alcatel. He completed his B. Eng in computer systems in 1993 from N.E.D. University of Engineering and Technology, Karachi, Pakistan, and completed his M. Eng in 1999 at Carleton University, Ottawa, Canada. He has published several papers and presentations at international conferences and seminars and is also engaged in IETF standardization activities.

Dedications

From Mobeen Tahir:

This book is dedicated to the memory of my father, Tahir Khan. He taught me how to take the first step in life.

To my wife, Sharmeen, and my kids, Mohammad and Iman, for their unconditional love.

To my mother, Sadiqa, and my siblings Noreen, Javaria, and Usman, for their prayers and support.

From Mark Ghattas:

This book is dedicated to my wife and son. I thank my wife, Amy, for her sacrifices, love, patience, and endless support to allow me to pursue my goals.

To my mom, Ehsan, who provided me opportunities, guidance, wisdom, and love, which made me the person, husband, and father I am today.

To my brothers, Matt and Paul, for the great technical discussions that last forever at the dinner table.

To Brian—our friendship keeps me inspired.

From Dawit Birhanu:

This book is dedicated to my wife, Lydia, and daughters, Leah and Blen, for their sacrifice, patience, love, and support. It is also dedicated to my mother, Negesu, and father, Birhanu, for their sacrifice and support to pursue my aspirations.

From Syed Natif Nawaz:

I dedicate this book in loving memory of my grandmother, Ameerunissa Begum, and to my mother, Haseena Begum, for all their sacrifices and support over the years and their love. I also dedicate this to my son, Taha, and my wife, Kouser Fathima, for filling my life with joy. To my sister, Arshiya Afshan, and brother-in-law, Shameeque. May their life be filled with joy and opportunities. Last but not the least, to my late father, Mr. Syed Yakoob Ali.

Acknowledgments

From Mobeen Tahir:

I would like to acknowledge the technical help given to me by several members of the IOS XR development community. I am particularly indebted to Pradosh Mohapatra, Brian Hennies, Muhammad Durrani, Arun Satyanayarana, Deepak Sreekanten, John Plunkett, Rakesh Gandhi, and Syed Kamran Raza, for answering my numerous questions and providing their expert advice. I would also like to point out the help and encouragement given to me by my colleague Muhammad Waris Sagheer.

From Mark Ghattas:

I would like to acknowledge Shahzad Burney and Waris Sagheer, who supported the conception and creativity of this project. A thank you to Anthony Lau, who helped me develop a "world" of experience with the multishelf platform. Thanks to Eddie Chami, Grant Socal, and Nikunj Vaidya for their input on best practices documents. I want to thank my co-authors Mobeen, Dawit, and Syed, who sacrificed personal and family time to meet commitments.

From Dawit Birhanu:

I would like to acknowledge the technical help given to me from several members of IOS XR and CRS development teams, and CRS deployment team. I would also like to acknowledge Lane Wigley, Ken Gray, Joel Obstfeld, and Yeva Byzek for their mentorship, inspiration, and support

Syed Natif Nawaz:

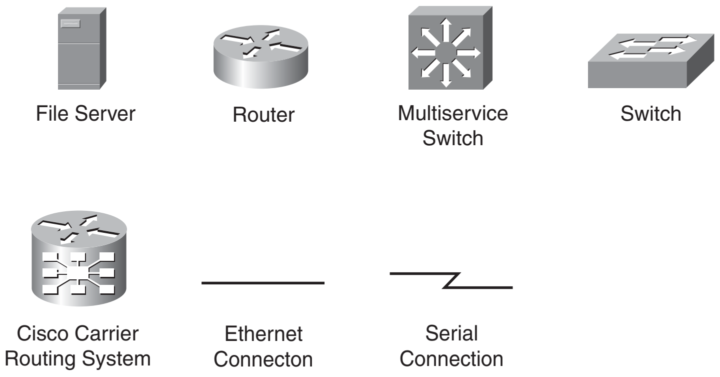
Thanks to all my co-authors for their effort and teamwork. Special thanks to my friends Waris and Shahzad during the inception of this book. I would also like to thank Jeffrey Liang and Lakshmi Sharma for helping me with their expertise and experience. I want to thank Kiran Rane, Srihari Sangli, Sai Ramamoorthy, Ravi Amanaganti, Pankaj Malhotra, and Paresh Shah for their unreserved support. As always I am grateful to my mom, Haseena Begum, my wife, Kouser Fathima, and sister, Arshiya Afshan, for being there for me.

The authors would like to send a special acknowledgement to Brett Bartow at Cisco Press, who has been ultra-supportive and understanding of the hurdles and delays we encountered. In addition, we thank Dayna Isley at Cisco Press, for her input and guidance supporting our content.

Contents at a Glance

Contents

Icons Used in This Book



Command Syntax Conventions

The conventions used to present command syntax in this book are the same conventions used in the IOS Command Reference. The Command Reference describes these conventions as follows:

- **Boldface** indicates commands and keywords that are entered literally as shown. In actual configuration examples and output (not general command syntax), boldface indicates commands that are manually input by the user (such as a **show** command).
- *Italic* indicates arguments for which you supply actual values.
- Vertical bars (|) separate alternative, mutually exclusive elements.
- Square brackets ([ ]) indicate an optional element.
- Braces ({ }) indicate a required choice.
- Braces within brackets ([{ }]) indicate a required choice within an optional element.

Foreword

Over the last several years, fiscal discipline has really dominated the industry. Both consumers and businesses expect far more from their communications providers than they did just a few years ago. Offering simple telephone dial tone and an Internet connection are not going to be enough for success. At the same time, however, service providers want to continue to reduce their operational costs. As a result, one of the main challenges telecommunications companies now face is to find ways to cost effectively bring innovative services to their customers. These drivers are why most providers are working on transitioning their disparate legacy networks to one, unified, converged network infrastructure based on IP combined with Multiprotocol Label Switching (MPLS). MPLS is a technology that translates various other telecommunications protocols, such as ATM or frame relay, so they can run over an IP-based network. By eliminating their multiple networks, service providers are greatly reducing their operational costs. And by moving to an IP/MPLS network, they can mix and match all communications types—voice, data, and video—into any service their customers might want.

We believe the CRS-1 will dramatically affect carriers and their capability to successfully transition to this new era in communications. Carriers worldwide are embracing convergence and almost unanimously agree that IP/MPLS is the foundation for their new infrastructures. The CRS-1 provides carriers the means to consolidate their networks in the most efficient and cost-effective way possible. Nothing on the market can match it in terms of scalability, reliability, and flexibility. It is a system that our service provider customers will be able to base their businesses on. And I firmly believe that carriers that deploy the CRS-1 will gain profound competitive advantage over their competition through operational efficiencies and service flexibility. As we like to point out, when service providers work with Cisco, they are not just working with a network equipment maker but, rather, a business partner.

Sameer Padhye
Sr. Vice President, Advanced Services
WW Service Provider Line of Business
Customer Advocacy

Introduction

This book is intended to provide a reference to users who plan or have implemented Cisco IOS XR software in the network. *Cisco IOS XR Fundamentals* provides an overview of IOS XR operation system infrastructure and hardware architecture on the Carrier Routing System. The intention of this book is to provide general networking topics in IOS XR that service providers may implement in the core network. It is not feasible to cover every aspect of IOS XR; however, the key configurations have been explained that are typically deployed in core networks.

Who Should Read This Book?

Readers who have a relatively strong working knowledge of Cisco IOS Software and routing protocols will benefit from the discussions and configuration examples presented.

How This Book Is Organized

Although this book could be read cover to cover, it is designed to provide a configuration overview on Cisco IOS XR to support implementation configuration and features in IOS XR. Chapter 1 provides an overview of the evolution of operating systems and an understanding of the underlying QNX operating system. Chapters 2 through 12 are the core chapters and can be covered in order. If you do intend to read them all, the order in the book is an excellent sequence to use.

Chapters 1 through 12 cover the following topics:

- **Chapter 1, "Introducing Cisco IOS XR":** This chapter discusses the evolution of network operating systems in service provider enviroments. It is important to understand the goals and requirement of service providers that influenced the goals of IOS XR.
- **Chapter 2, "Cisco IOS XR Infrastructure":** This chapter discusses the interworkings of IOS XR. It helps you understand IOS XR microkernel architecture, process scheduling, interprocess communications, system database, and distributed services.
- **Chapter 3, "Installing Cisco IOS XR":** This chapter discusses various procedures for installing IOS XR on the Carrier Routing System.
- **Chapter 4, "Configuration Management":** This chapter provides a deeper insight into how IOS XR is different when configuring interfaces, out of band management, and features such as rollback and commit commands. Understanding these features will help you better manage the system.
- **Chapter 5, "Cisco IOS XR Monitoring and Operations":** This chapter explores how monitoring works in IOS XR. As IOS XR operates as a real-time operating system, there are monitoring tools that provide deeper inspection of activities on the system.
- **Chapter 6, "Cisco IOS XR Security":** This chapter examines inherent policers that provide a layer of security within the operating system. The importance of Local Packet Transport System (LPTS) is discussed.

- **Chapter 7, "Routing IGP":** This chapter covers the basics of routing protocol configurations. It provides configuration examples to show how IGP features are configured in IOS XR.
- **Chapter 8, "Implementing BGP in Cisco IOS XR":** This chapter introduces the IOS XR implementation of BGP. This chapter assumes that you have prior experince and knowledge of the BGP protocol and focuses on unique aspects of IOS XR BGP configuration. This chapter also provides details on Routing Policy Language as a vehicle for implementing BGP routing policies.
- **Chapter 9, "Cisco IOS XR MPLS Architecture":** This chapter discusses Multiprotocol Label Switching (MPLS), an important technology for building converged network infrastructure and services. This chapter assumes that you are familiar with MPLS protocols and operations. This chapter discusses IOS XR MPLS architecture, features, implementation, and configuration. It covers LDP, Layer 3 VPN, VPWS, VPLS, and MPLS Traffic Engineering.
- **Chapter 10, "Cisco IOS XR Multicast":** This chapter discusses when to use queuing and which queuing technique to use. This chapter also examines Weighted Fair Queuing (WFQ), Custom Queuing, and Priority Queuing and addresses the need for compression in today's enterprise network.
- **Chapter 11, "Secure Domain Router":** This chapter covers the concept of SDRs. It discusses the Distributed Route Processor (DRP) hardware needed to implement SDRs and provides configuration examples.
- **Chapter 12, "Understanding CRS-1 Multishelf":** This chapter discusses the Cisco implementation of the CRS-1 multishelf system. The key components are discussed to understand the architecture and troubleshooting of a CRS-1 multishelf system. A fabric troubleshooting section is covered to support implementation and operation.

This chapter covers the following topics:

- Evolution of Networking
- Requirements for Carrier-Grade NOS
- Operating System Concepts
- High-Level Overview of Cisco IOS XR
- Cisco IOS XR Platforms
- References

This chapter reviews the evolution of network operating systems (NOS), requirements for current and future networks, and how Cisco IOS XR meets these requirements. The first section of this chapter provides an overview of the evolution of networking. The second section outlines the requirements for a carrier-grade NOS that underpins a converged network with critical applications. The third section reviews basic concepts of operating systems. The final sections provide a high-level overview of Cisco IOS XR.

CHAPTER 1

Introducing Cisco IOS XR

Evolution of Networking

In the 1980s the main network applications were limited to e-mail, web, file, printer, and database. Silicon technology for hardware (HW)-based packet forwarding was not yet fully developed, and transmission speed, CPU power, and memory capacity were very limited. As a result, routers and the underlying NOS were primarily designed to efficiently use CPU and memory resources for packet forwarding. A *NOS* is an operating system that is specifically designed for implementing networking and internetworking capabilities. Network devices such as routers and switches are empowered by a NOS.

Moreover, in the early days of data networking there was a plethora of competing networking protocols in addition to Internet Protocol (IP). Some became industry standards and others remained proprietary. Table 1-1 shows the protocols at different OSI layers that were once prevalent to varying degrees.

Table 1-1 *Protocols That Were in Use in the Early Days of Data Networking*

Protocols	OSI Layer
Token Ring, Fiber Distributed Data Interface (FDDI), Switched Multimegabit Data Service (SMDS)	1–2
ATM, Frame Relay	2–3
Internetwork Packet eXchange (IPX), International Standards Organization ConnectionLess Network Services (ISO CLNS), AppleTalk, DECNet, Xerox Network Services (XNS), IBM System Network Architecture (SNA), Apollo Domain, Banyan Virtual Integrated Network Services (VINES)	3

Routers were designed to support a variety of multiple protocols including IP, Ethernet, SONET/SDH, and some of the protocols shown in Table 1-1.

Network operators had several service-specific networks, each managed and operated by a different team. It was not uncommon for a service provider to maintain a separate PSTN network for telephony, an ATM data network, a Frame Relay data network, a public data network for Internet customers, a separate network for mobile backhaul, and a transport network to support all services. Some network operators still have a legacy of multiple networks; however, they are actively migrating to a converged network.

Although networking services such as e-mail, web browsing, file transfer, instant messaging, VoIP, and so on are taken for granted today, they were either nonexistent or considered privileged services for a few users at large enterprise, academic, and government institutions.

Over the past few decades the network, users, and services have evolved dramatically as follows:

- **Applications:** In the 1980s there were just a few network applications, namely e-mail, file, database, and print services. Today there are countless applications, including video conferencing, instant messaging, IPTV, telepresence, telemedicine, peer-to-peer sharing, video surveillance, online banking, online shopping, and so on.
- **User size:** Until the mid-1990s, data networking usage was limited to large enterprise, government, and academic institutions for limited applications. Based on data from Internet World Stats, Internet usage has grown from 16 million users in 1995 to 1.46 billion users in 2008. Moreover, per-capita bandwidth usage has increased dramatically since the mid 1990s.
- **Transmission capacity:** Transmission capacity of a single fiber pair has increased from 155Mbps in the early 1990s to multi-terabits today (realized with dense wavelength division multiplexing [DWDM] technologies). The Trans-Pacific Express (TPE) submarine cable that connects the United States to mainland China has an initial capacity of 1.28 terabits per second with a designed maximum capacity of 5.12 terabits per second.
- **Processing and memory capacity:** CPU speed and complexity increased from tens of megahertz single core processors in the early 1990s to multigigahertz multicore processors in 2009 following Moore's law. Memory capacity and access speed have seen similar growth—from a few megabytes of memory capacity in the early 1990s to many gigabytes in 2009. Moore's law, which is named after Intel co-founder Gordon E. Moore, states that processor and memory capacity doubles approximately every two years.
- **Protocols:** From several protocols in the early 1990s (as shown in Table 1-1), the network has consolidated toward IPv4/IPv6 and Ethernet protocols.
- **Networks:** Network operators have migrated or are in the process of migrating from multiple networks, each dedicated for specific function to a single converged network capable of supporting multiple services.

Requirements for Carrier-Grade NOS

Service providers are striving to provide solutions that can sufficiently satisfy the needs of their customers. Businesses are demanding integrated data, voice, video, and mobility services with high availability, security, and fast provisioning. Consumers want broadband access with bundled service of voice, video, mobile wireless, and data on a single bill. Governments are pushing for broadband access to every home and a resilient infrastructure that can survive catastrophic failures.

This section describes the requirements that a carrier-grade NOS needs to satisfy to meet the requirements of network operators.

Convergence

A carrier-grade NOS should have the capability to enable infrastructure and service convergence. Network convergence is critical to lowering capital and operational expenditure. Service convergence is vital to meeting customer demands and to offer new revenue-generating services.

Scalability

A converged network infrastructure should be able to scale seamlessly with respect to control plane, data plane, and management plane without interruption to existing services. The growth of customers, access bandwidth, and traffic volume per customer every year is pushing the scalability demand on every aspect of the network infrastructure. To cope with growth, the network operator might have to add additional hardware in the form of network ports, transport links, line cards, route processing cards, power modules or chassis in a multi-chassis system. The NOS should be able to support the addition of different system components without service disruption.

Availability

In a converged network, routers are carrying critical traffic including voice, emergency service traffic, video broadcasting, video conferencing, and business-critical data with availability requirement of 99.999% or better. To achieve carrier-grade availability requirements, a network operating system should be able to support a number of high availability features as described in this section.

Hardware Redundancy

Although it is possible to reduce the probability of hardware failure, it is virtually impossible and cost-prohibitive to reduce it to zero. Therefore, to achieve carrier-grade availability it is important to build the system with redundant hardware modules—particularly for system-critical subsystems. In addition, the NOS should have the necessary software capability to enable the system to operate with no or minimal service disruption when such a module fails, and when it is subsequently removed, upgraded, or replaced.

Failure Recovery and Microkernel-Based NOS

Modern operating systems and applications are complex, and are developed by hundreds of software engineers. It is virtually impossible to have defect-free operating systems. A software component might fail not only due to software defect but also due to memory corruption and malicious attacks. A carrier-grade NOS should be able to contain and recover from most software failures without service disruption.

Modern operating systems have kernel and nonkernel components. In general, a failure in a nonkernel software component will not impact the kernel or other nonkernel components. A kernel failure, however, will cause system reload. This suggests that it is important to keep most software components outside the kernel and to keep only minimal functionality in the kernel. This type of operating system is called a *microkernel-based operating* *system*. Multitasking, multithreading, and memory protection, which are discussed in the next section, are also critical components of a carrier-grade NOS.

Process Restartability

When a software process fails, the operating system should be able to restart the process. When a process is restarted, it should be able to recover its state so that it can seamlessly continue its functions without disrupting service. This capability is referred to as *process restartability*.

Failure Detection

The network operating system should also support network features that enable quick failure detection and rerouting of traffic around failed links, modules, or routers.

Software Upgrades and Patching

Carrier-grade NOS should support software upgrade and/or patching with no or minimal disruption to service. It is important that it has software patching capability to apply critical software updates and minimize frequent full software upgrades.

Security

A router has two primary security functions:

- To protect customer and service provider infrastructure by supporting network security features such as unicast reverse path forwarding (uRPF), access control list (ACL)–based filtering, and prefix filtering
- To protect the router from malicious or unintended security attacks and intrusions, which is the primary focus here

The operating system must provide effective mechanisms to protect the routing protocols from malicious attacks. It should also provide granular access control to protect the router from unauthorized access. Distributed denial of service (DDoS) attacks are common and becoming sophisticated. The NOS should minimize the impact to data, control, and management plane functions due to such attacks.

Service Flexibility

Carriers are demanding a routing system that has a long life cycle. This requires that the addition of new services should not require a fork-lift upgrade. Carrier-grade NOS needs to support the addition of new software features, line cards, and/or service modules with no or minimal service disruption. This can be achieved with modular software packaging, the support of service modules, and partitioning of systems into multiple routing domains.

Operating System Concepts

Computer systems, including "embedded" systems such as routers, have an operating system that is responsible for providing a number of services to the applications. Coordination of processing activities and access to hardware resources such as memory, network interfaces, and disk are also essential functions provided by an operating system. Figure 1-1 shows the relationships among the operating system, applications, and hardware resources.

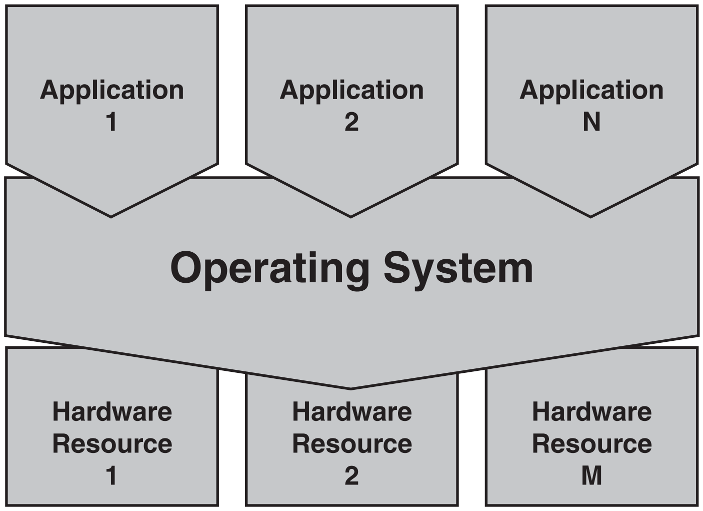


Figure 1-1 *Operating System Interaction with Hardware and Applications*

Basic Functions of an Operating System

Operating systems provide a number of services to applications. The basic functions offered by an operating system include process scheduling, interrupt handling, memory management, interprocess communication, and common routines (or library). These basic functions of OS are discussed in more detail in this section.

Process Scheduling

A *process* is a software program execution instance running on a system that has the capability to execute multiple program instances. Multiple processes can be spawned simultaneously from a single program. In a multitasking operating system multiple processes can time-share CPU resources, giving the user a perception of simultaneous processing. In a multitasking system, a process might have to relinquish control of the CPU before it completes the execution of its current task.

In a *cooperative multitasking* system, a process voluntarily relinquishes control only after completing execution or while waiting for an event. This could result in CPU starvation of other processes while waiting for the current process to relinquish control.

On the other hand, in a *preemptive multitasking* system a currently running process might be forced to relinquish control of the CPU. This is called *preemption*, and it can occur when either a high-priority process becomes ready or after the current process has run for the time allocated to it.

When the operating system preempts a process it is necessary to preserve the state of the process before relinquishing control to another process so that it can resume its execution when it gets to run again. *Context switching* is the mechanism by which a processes state is saved when it is preempted and retrieved when it resumes execution.

Most modern operating systems support running multiple instances of the same process concurrently. These types of operating systems are known as *multithreaded*. A *thread* is the smallest unit of execution within a process.

Interrupt Handling

Interrupt is a signal from hardware or software indicating a need for immediate attention. It causes the operating system to suspend a currently running process and dispatch an interrupt handling routine or process. A running process can also execute an interrupt instruction and trigger context switch to an interrupt handler. When executing a critical routine, the operating system can inhibit certain interrupts until the critical routine is completed. This is known as *interrupt masking*.

Memory Management

The operating system is responsible for managing the entire system memory, including allocation of memory to processes and ensuring that a process does not corrupt memory that belongs to another process. *Memory protection* is a mechanism by which a process is prevented from accessing memory locations other than the memory space allocated to it. With memory protection, each process runs in its own memory space. A defect in one process or a malicious attack to one process will not impact other processes.

In operating systems that support memory protection, some forms of communication between processes are better handled using *shared memory*, which is accessible by multiple processes. The operating system provides different synchronization mechanisms between processes that are writing to or reading from shared memory regions.

In a *monolithic operating system*, all processes share the same address space and the system does not provide fault isolation among processes. A monolithic system can offer better utilization of CPU cycles because it has lower overhead with respect to memory access, interprocess communication, and context switching. It might be useful in scenarios in which CPU resources are expensive and the overall system is simple with small code size.

In operating systems that support memory protection, the OS process that is responsible for managing other processes, memory, and other system resources is known as the *kernel*, and the OS is often referred to as *kernel-based OS*. The kernel can also contain other services, depending on the implementation. It runs in a separate memory space from the rest of the system and is protected from memory corruptions caused by other processes outside the kernel.

A failure in a nonkernel process does not impact the kernel and other processes. However, a failure in the kernel processes impacts all applications. In a *microkernel* system, only essential core OS services reside inside the kernel. All other services, including device drivers and network drivers, reside in their own address space. This has important resilience implications in that a failure in a device or network driver is self-contained and does not propagate to the kernel or other applications. Device and network drivers can also be restarted without restarting the whole system.

Synchronization

When multiple applications are running concurrently and attempt to access a resource such as disk drive, it is important to make sure that data integrity is preserved and resource is allocated fairly. There are different mechanisms that network operating systems provide to synchronize events and resource access.

Interprocess Communication

The operating system provides the interprocess communication (IPC) mechanism for processes running in separate address spaces because they cannot use the memory to exchange data. IPC communication can also occur between processes running on the main route processor and the processes running on different components in the device, including line cards and power supplies.

Dynamic Link Library

It is common for multiple applications to use a set of common routines. When these applications are running in separate protected memory address spaces, the common routines have to be duplicated in each address space, which is a waste of memory space. To avoid this problem, operating systems provide a mechanism to share common routines. This mechanism is called *dynamic linked library (DLL)* or *Libc* (C standard library). This allows the OS to load only active libraries into device memory and enables different processes to share the same libraries. This is a robust fault containment and software modularization mechanism. It also allows the sharing of common code among different applications.

Portable Operating System Interface

Portable Operating System Interface (POSIX) is a set of IEEE specifications that define kernel APIs, thread interfaces, kernel utilities, and more. POSIX also defines a conformance test suite. If an operating system passes the test suite, it is called a POSIX-conforming OS. An OS that adheres to POSIX compliance is considered highly flexible and provides maximum portability for additional features or application development. An application program developed for one POSIX-compliant OS can easily be ported with minimal effort to another POSIX-compliant OS.

High-Level Overview of Cisco IOS XR

As the world is becoming increasingly dependent on IP-based network infrastructure, network operators are demanding a high degree of reliability and availability. Cisco IOS XR Software is designed to meet the stringent requirements of network operators. It is designed to provide the following:

- A high level of scalability
- Distributed forwarding architecture
- Exceptionally high reliability and resiliency
- Service separation and flexibility
- Robust security

- Modularity across all software components
- Hierarchical configuration and robust configuration management
- Better manageability

Cisco IOS XR software is a highly distributed, secure, highly modular, and massively scalable network operating system that allows uninterrupted system operation. It is a microkernel-based operating system with preemptive multitasking, memory protection, and fast context switching. The microkernel provides basic operating system functionalities including memory management, task scheduling, synchronization services, context switching, and interprocess communication (IPC).

The microkernel used in Cisco IOS XR is QNX Neutrino real-time operating system (RTOS) from QNX Software Systems. The kernel is lightweight and does not include system services such as device drivers, file systems, and network stack. Figure 1-2 shows the IOS XR microkernel architecture.

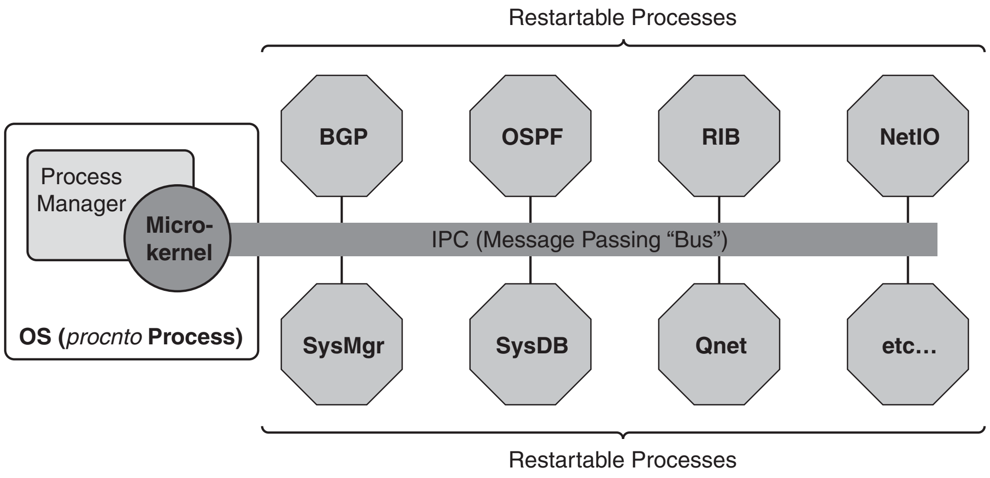


Figure 1-2 *Cisco IOS XR Microkernel Architecture*

All processes outside the microkernel (procnto) are individually restartable. If any of the processes, including SysMgr, SysDB, Qnet, or BGP, is restarted it does not cause the entire system to reload. When a process restarts, it recovers its states from persistent storage or peer processes, also called *collaborators*. For example, if the Routing Information Base (RIB) process restarts it will restore the RIB table from its collaborators, which are routing protocol processes such as OSPF, BGP, IS-IS, and so on. As a result, the RIB table is rebuilt and there is no traffic disruption if the RIB process is restarted.

Cisco IOS XR employs two distribution models to achieve higher performance and scalability. The first distribution model uses localization, which performs processing and storage closer to the resource. With this model, a database specific to a node is located on that node. Also processes are placed on a node where they have greater interaction with the resource. For example, Address Resolution Protocol (ARP), interface manager (IM), Bidirectional Failure Detection (BFD), adjacency manager, and Forwarding Information Base (FIB) manager are located on the line cards and are responsible only for managing resources and tables on that line card. System databases specific to the line card, such as interface-related configurations, interface states, and so on, are stored on the line card. This enables IOS XR to achieve faster processing and greater scalability.

The second distribution model uses load distribution in which additional route processors (RPs or distributed RPs [DRP]) are added to the system and processes are distributed across different RP and/or DRP modules. Routing protocols, management entities, and system processes are examples of processes that can be distributed using this model. For example, we can classify the processes into three groups as follows and allocate each group to run on one RP or active/standby RP pair:

- **Group 1:** All routing protocols or processes, including BGP, ISIS, LDP, RSVP, PIM, MSDP, and RIB
- **Group 2:** All management entities, including SNMP server, SSH, Telnet, XML, and HTTP
- **Group 3:** All other processes

This model enables the operator to add additional RPs or DRPs in the system as needed to offload processing from one RP to another, essentially increasing the overall processing power of the system.

Cisco IOS XR provides a clear separation of management, control, and data plane. Figure 1-3 illustrates the IOS XR architecture and the separation of the management, control, and data planes.

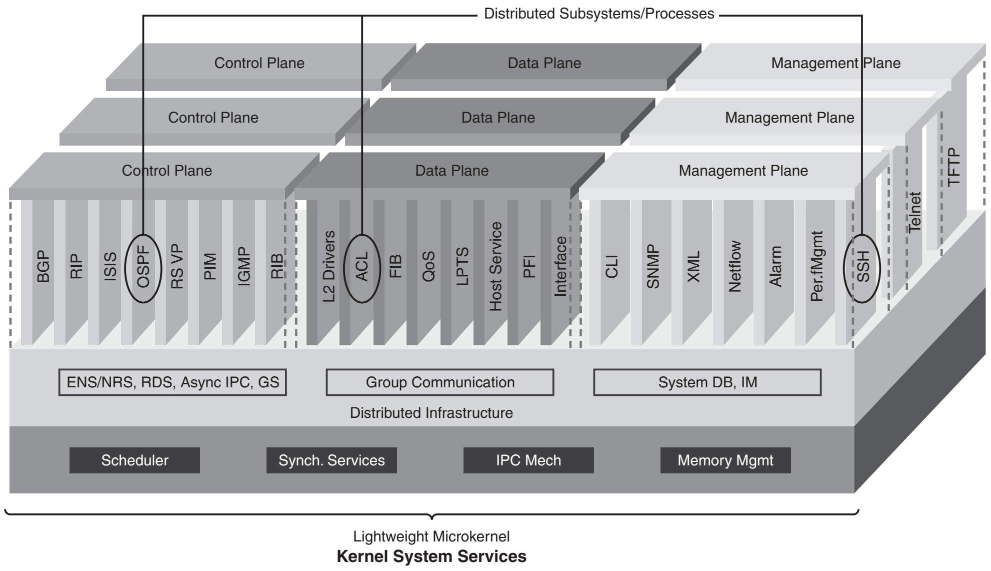


Figure 1-3 *Cisco IOS XR Architecture: Separation of Management, Control, and Data Planes*

Each routing control plane or management plane process runs on one or multiple route processors (RP) and/or distributed RP nodes. Data plane processes are located on each node that participates in packet forwarding, including RP and line card.

Cisco IOS XR supports partitioning of a system into multiple secure domain routers (SDR) at physical boundaries. SDRs share only chassis, power supply, fan tray, and related system components. Each line card or RP belongs to only one SDR. Cisco IOS XR SDRs provide fault and security isolation because they are defined at physical boundaries. A fault, resource starvation, or security breach on one SDR does not impact other SDRs in the same system. An SDR can be defined with just one RP, but it can have multiple RPs and LCs.

Figure 1-4 shows a system partitioned into three SDRs: default SDR, SDR 1, and SDR 2. The SDR that has the designated shelf controller (DSC) is the default SDR. DSC is the main RP (or RP pair for redundancy) on the system.

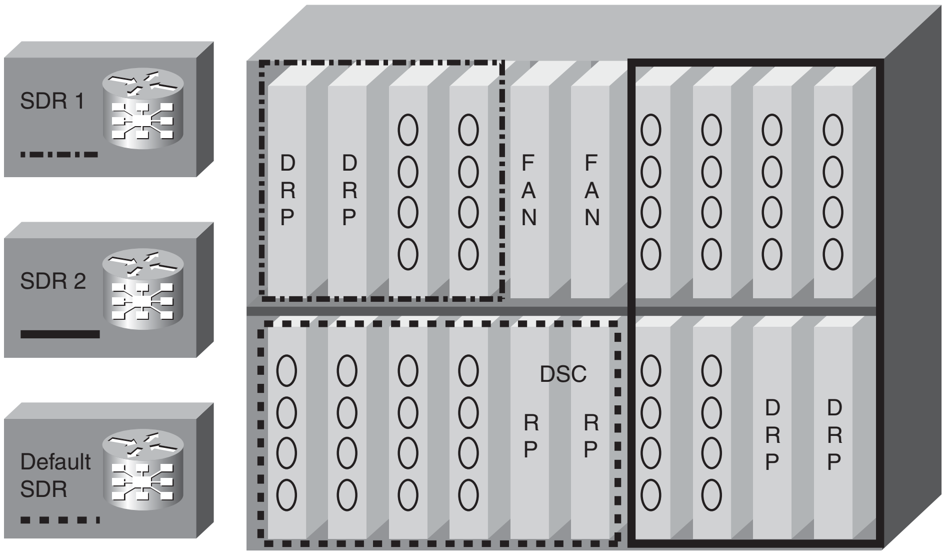


Figure 1-4 *Partitioning System into Secure Domain Routers*

Cisco IOS XR uses a two-stage fully distributed forwarding architecture. Each line card has forwarding information base (FIB) and local adjacency information base (AIB) for local interfaces on that line card. When a packet first enters the system, the ingress line card performs ingress feature processing and FIB lookup. The FIB lookup returns sufficient information for the ingress line card to deliver the packet to the appropriate egress line cards. The ingress line card does not need to know the full adjacency information of the egress interface. The ingress line card sends the packet through the fabric to the egress line card. The egress line card performs egress feature processing and FIB lookup to get full adjacency and layer 2 rewrite information. The packet is then sent to the outbound interface with an appropriate layer 2 header.

The purpose of two-stage forwarding is to get better scalability and performance. This is critical because Cisco IOS XR is designed to achieve a very high degree of scalability in different dimensions, including bandwidth capacity, number of routes, and number of customer connections.

In Cisco IOS XR, all transit traffic is processed in HW and does not involve any LC or RP CPU processing. Only traffic destined to the router or originating from the router is processed by LC or RP CPU. Cisco has developed an innovative processing and delivery mechanism for packets destined to the router. This mechanism is called *local packet transport service (LPTS)*. If a packet enters the system and FIB lookup in HW determines that the packet needs to be delivered to the local system, it will be handed over to LPTS process for additional HW processing. LPTS determines what application it is destined to and sends the packet to the node where the application resides. For example, if a BGP packet is received, the ingress LC will send it directly to the RP where the BGP process is located. The HW forwarding engine on the LC sends the packet through the fabric to the RP. The LC CPU does not touch this packet.

Cisco IOS XR LPTS also acts as a dynamic integral firewall and protects the system from denial of service and other forms of attacks. To protect the system from DoS attacks, it monitors and polices the traffic destined to the router. For example, BGP or any other type of control packets destined to the RP must conform to the policing thresholds set by the LPTS process. In case of BGP, the policer value is set such that regular BGP updates are not impacted. However, if someone maliciously sends a large amount of BGP updates, LPTS protects the RP CPU from being overwhelmed with bogus BGP packets. The policer value also depends on the status of the BGP session for which the packet is sent. If the packet belongs to a configured neighbor and the session is not yet established, the rate will be lower. On the other hand, if the packet matches an established session the rate will be higher. Note that it is very hard to generate bogus BGP packets belonging to an established session because the attacker must know the source and destination port of the BGP session in addition to the source and destination IP addresses.

LPTS does not require user configuration—it is enabled by default and updated dynamically as the system is configured and sessions come up and down. The LPTS policer values, however, are user configurable.

Cisco IOS XR Platforms

This section provides a brief overview of Cisco IOS XR–based platforms. It is not intended to provide a detailed systems architecture for these platforms. Visit the Cisco website (http://www.cisco.com/) to get detailed information on each of the platforms described in this section.

Cisco CRS-1 Carrier Routing System

Cisco CRS-1 is the first platform to run IOS XR. It is designed for high system availability, scale, and uninterrupted system operation. CRS-1 is designed to operate either as a single-chassis or multichassis system. It has two major elements: line card chassis (LCC) and fabric card chassis (FCC). Details about each system follow:

- **CRS-1 16-Slot Single-Chassis System** is a 16-slot LCC with total switching capacity of 1.2 Tbps and featuring a midplane design. It has 16 line card and 2 route processor slots.
- **CRS-1 8-Slot Single-Shelf System** is an eight-slot line card chassis with total switching capacity of 640 Gbps and featuring a midplane design. It has eight line card and two route processor slots.
- **CRS-1 4-Slot Single-Shelf System** is a four-slot line card shelf with total switching capacity of 320 Gbps. It has four line card and two route processor slots.
- **CRS-1 Multi-Shelf System** consists of 2 to 72 16-slot LCC and 1 to 8 FCC with a total switching capacity of up to 92 Tbps. The LCCs are connected only to the FCCs where stage 2 of the three-stage fabric switching is performed. The FCC is a 24-slot system.

Cisco XR 12000 Series

Cisco XR 12000 series is capable of a 2.5 Gbps, 10 Gbps, or 40 Gbps per slot system with four different form factors:

- **Cisco 12016, Cisco 12416, and Cisco 12816** are full-rack, 16-slot, and 2.5-, 10- and 40-Gbps per slot systems, respectively.
- **Cisco 12010, Cisco 12410, and Cisco 12810** are half-rack, 10-slot, and 2.5-, 10- and 40-Gbps per slot systems, respectively.
- **Cisco 12006 and Cisco 12406** are 1/4-rack, 6-slot, and 2.5- and 10-Gbps per slot systems, respectively.
- **Cisco 12404** is a four-slot, 10-Gbps per slot system.

Cisco ASR 9000 Series

ASR 9000 Series Aggregation Service Router is targeted for carrier Ethernet services and delivers a high degree of performance and scalability. It can scale up to 6.4 Tbps per system. It comes with two form factors:

- **Cisco ASR 9010** is a 10-slot, 21-rack unit (RU) system.
- **Cisco ASR 9006** is a 6-slot, 10-rack unit (RU) system.

Summary

Networking has evolved from limited use for specialized applications using several disparate networks to a critical infrastructure that is relied on by businesses, public services, government, and individuals for an increasing number of applications. As a result, network operators are demanding a very high degree of availability, reliability, and security for the routers that constitute their network infrastructure. IOS XR is designed to meet this challenge.

Cisco IOS XR is a microkernel-based operating system with preemptive multitasking, memory protection, a high degree of modularity, and fast context-switching capabilities. Because each process outside the microkernel is restartable without impacting the rest of the system, failure of a process due to memory corruption of software defect does not impact other parts of the system.

To achieve a high degree of scalability and performance, Cisco IOS XR employs two forms of distribution: localization and load distribution. Localization refers to performing processing and storage closer to the resource. *Load distribution* refers to offloading of processing from one RP to another with the objective of increasing overall processing power of the system.

Cisco IOS XR uses a two-stage fully distributed forwarding architecture. When a packet first enters the system the ingress linecard performs ingress feature processing and FIB lookup. The FIB lookup returns sufficient information for the ingress line card to deliver the packet to the appropriate egress line cards. The egress line card performs egress feature processing and FIB lookup to get the full L2 adjacency information.

References

- **Internet World Stats.** http://www.internetworldstats.com/
- **Cisco.** Cisco IOS XR Configuration Guides. http://www.cisco.com/

This chapter covers the following topics:

- Cisco IOS XR Kernel
- Cisco IOS XR System Manager
- Interprocess Communication
- Distributed Services
- Process Placement
- Cisco IOS XR System Database
- High Availability Architecture
- Forwarding Path
- References

Cisco IOS XR is designed for massively scalable systems with particular focus on continuous system operation, scalability, security, and performance. This chapter discusses the IOS XR infrastructure and how it achieves the stated goals of IOS XR. The first section discusses the microkernel used by IOS XR. Subsequent sections discuss interprocess communication (IPC), IOS XR System Database, distributed system services, process management, and high availability.

CHAPTER 2

Cisco IOS XR Infrastructure

Cisco IOS XR Kernel

Cisco IOS XR is a highly distributed microkernel-based network operating system. The microkernel used by Cisco IOS XR is QNX Neutrino real-time operating system (RTOS), which is from QNX Systems. The microkernel is lightweight and provides only a few fundamental services. It is responsible for interrupt handling, scheduling, task switching, memory management, synchronization, and interprocess communication. The microkernel does not include other system services such as device drivers, file system, and network stacks; those services are implemented as independent processes outside the kernel, and they can be restarted like any other application.

The microkernel is a POSIX-compliant kernel. POSIX defines OS specifications and test suites for APIs and OS services that a POSIX-compliant OS has to implement. Applications and services developed for a POSIX-compliant kernel can easily be ported to another POSIX-compliant kernel. If the need arises in the future, Cisco IOS XR can easily be ported to another POSIX-compliant OS.

The essential aspect of a microkernel-based OS is modularity. The microkernel provides a very high degree of modularity. The OS is implemented as a team of cooperative processes managed by the microkernel and glued by its message-passing service. Each process is running in its own address space and is protected from memory corruption of other processes. An important aspect of microkernel architecture is its fast context switching capability, which provides the impetus to a high degree of modularity. Because the CPU cost associated with context switching is minimal, it provides greater incentive to implement each application and service as its own process and in its own memory address space. For example, Cisco IOS XR implements BGP, OSPF, OSPFv3, RIBv4, RIBv6, and so on as separate processes. Moreover, if multiple OSPF processes are configured on the router each one will be assigned its own process instance completely separate from other OSPF processes. This greater modularity is made possible due to the fast context-switching capability of the microkernel and efficient interprocess communication provided by QNX and enhanced by Cisco. Interprocess communication is discussed in greater detail in the section "Interprocess Communication," later in this chapter.

Threads

As illustrated in Figure 2-1, the OS is a group of cooperating processes managed by a small microkernel. The microkernel provides thread scheduling, preemption, and synchronization services to the processes. It also serves as a message-passing "bus." The microkernel and the process manager together form the procnto process. Each process runs in its own address space and can be restarted without impacting other processes.

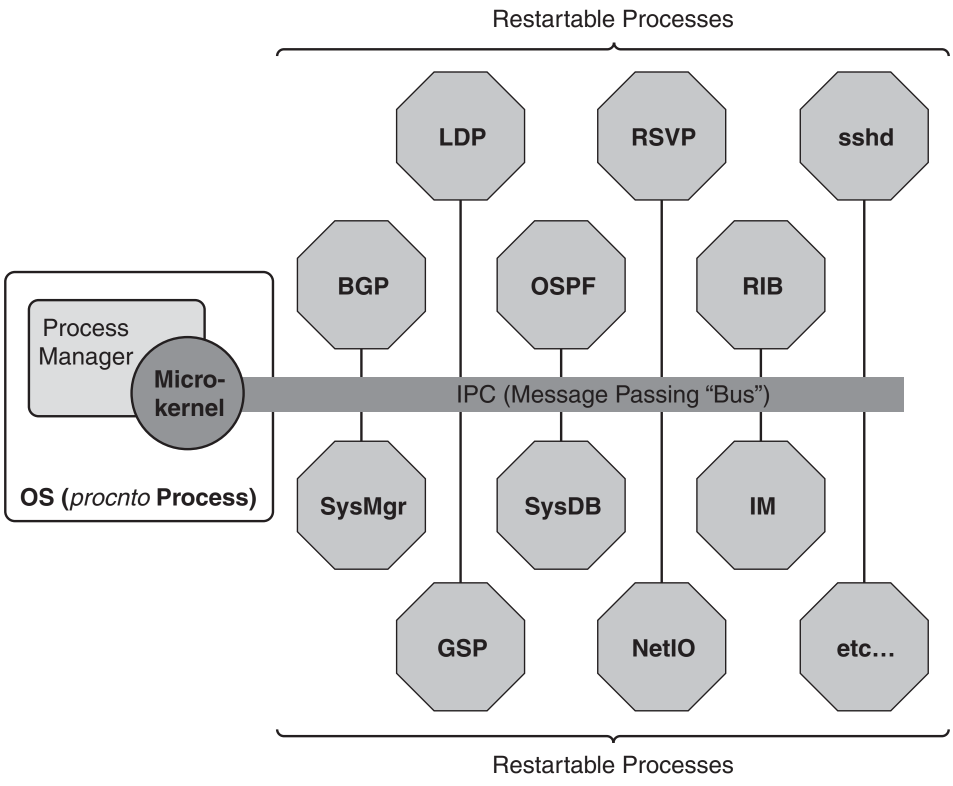


Figure 2-1 *Microkernel-Based Operating System*

When developing an application, it is often desirable to execute several algorithms concurrently. This concurrency is achieved using multiple threads within a process. A *thread* is the minimum unit of execution and scheduling. A *process*, on the other hand, is a container for related threads and defines the memory address space within which the threads can execute. There is at least one thread per process. Threads are discussed in more detail in the section "Cisco IOS XR System Manager."

For example, as you can see from the **show processes threadname** 120 output in Example 2-1, the BGP process in IOS XR has several threads that each perform a specific task, including input, output, import, and so on. In the following sample output, 120 is the jobid of BGP process. Jobid (JID) is a unique number assigned to each process, and it is covered in more detail in the section "Cisco IOS XR System Manager" later in this chapter.

Example 2-1 *Thread Names for the BGP Process*

```
RP/0/RP0/CPU0:CRS-A#show processes threadname 120
! 120 is the jobid of bgp process
JID    TID  ThreadName       pri   state     TimeInState         NAME
120    1    io-control       10   Receive        0:00:04:0166   bgp
120    2    chkpt_evm        10   Receive       96:23:26:0941   bgp
120    3    label-thread     10   Receive        0:00:16:0525   bgp
120    4    rib-update ID 0  10   Receive        0:00:16:0522   bgp
120    5    async            10   Receive       50:49:09:0707   bgp
120    6    io-read          10   Receive        0:01:16:0534   bgp
120    7    io-write         10   Receive        0:00:16:0532   bgp
120    8    router           10   Receive        0:00:16:0533   bgp
120    9    import           10   Receive        0:00:16:0529   bgp
120    10   update-gen       10   Receive        0:00:16:0529   bgp
120    11   crit-event       10   Receive        0:00:16:0525   bgp
120    12   event            10   Receive        0:00:32:0777   bgp
120    13   management       10   Receive        0:00:16:0549   bgp
120    14   rib-update ID 1  10   Receive        0:00:55:0617   bgp
RP/0/RP0/CPU0:CRS-A#
```

Figure 2-2 shows the most common thread states and transitions between the states. The inner circle actually represents two distinct states: ready and running. A thread state can transition from ready to running and vice versa. A thread in running state may also transition to any of the other states shown in Figure 2-2.

Cisco IOS XR microkernel uses a preemptive, priority based, and non-adaptive scheduling algorithm. Each thread is assigned a priority. The scheduler is responsible for selecting the next thread to run based on the priority assigned. The highest priority thread in ready state is selected to run. There is a ready state first in, first out (FIFO) queue for each priority level.

The idle thread is a special thread of the procnto process in that it is the only thread that runs at priority 0 and uses FIFO scheduling. Also, it is either in running or ready state and it never relinquishes CPU voluntarily. However, because it uses the lowest priority, it can be preempted by any other process that is in ready state.

A running thread may be moved to a different state due to system call (such as a kernel call, exception, or hardware interrupt), getting blocked, preempted, or voluntarily yielding. If a running thread is preempted by a higher priority thread, it moves to the head of the ready queue for its priority. On the other hand, if it is preempted after consuming its timeslice or it voluntarily yields the process, it moves to the end of the ready queue for its priority. *Timeslice* is the maximum time that a running thread can consume while one or more threads are in the ready queue for the same priority level as the running thread.

A running thread blocks when it needs to wait for an event to occur such as a reply message. When a thread is blocked it moves to the corresponding blocked state and stays there until it is unblocked. When the process is unblocked, it normally moves to the tail of the ready queue for its priority. There are some exceptions to this rule.

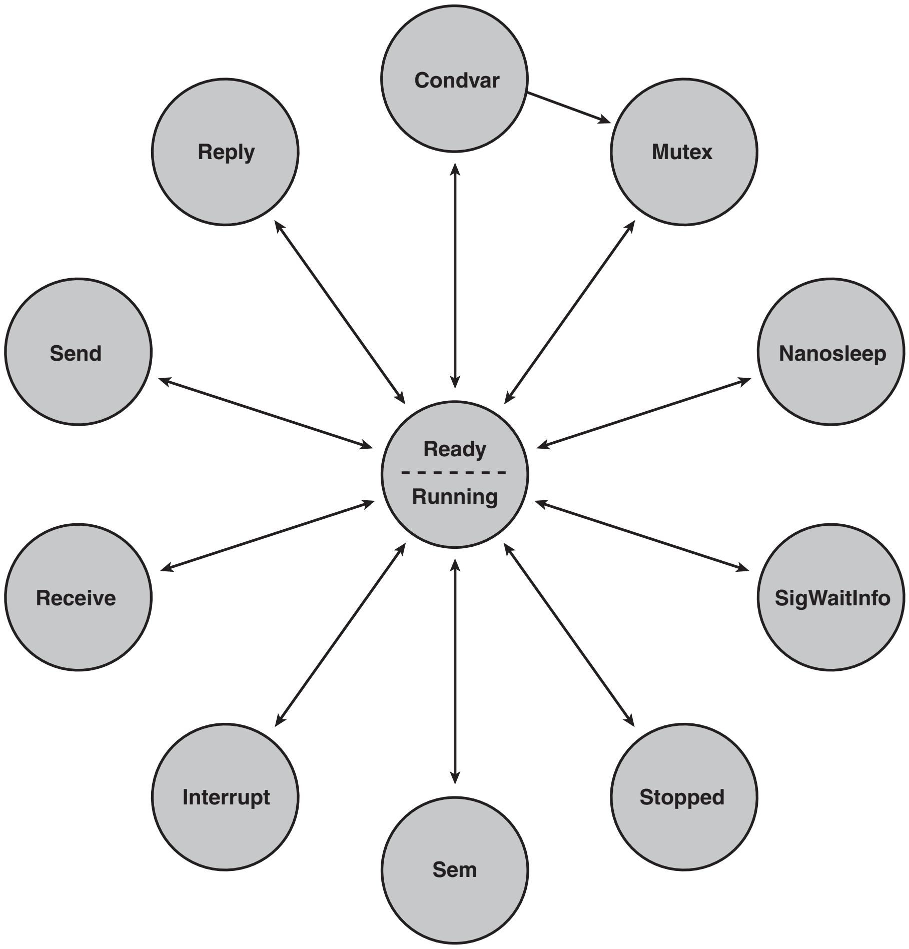


Figure 2-2 *Most Common Thread States and Transitions*

For example, if a server thread is waiting for a client request, it is in a receive blocked state. Suppose the blocked server thread has priority 10 and is unblocked by a client thread at priority 20 sending a request and waiting for a reply. This will unblock the server thread and move the client thread to reply blocked state. If the server thread is moved to the ready queue for priority 10 and there are several threads in ready state at priority 15, it will impact the response time for the client even though the client thread has priority 20. This problem is known as priority inversion. To prevent priority inversion, the microkernel uses priority inheritance, which temporarily boosts the priority of the server thread to match that of the client thread (20) and places the server at the ready queue for the client's priority (20).

Scheduling Algorithms

The microkernel provides the following three scheduling algorithms to meet needs for different scenarios:

- FIFO scheduling
- Round-robin scheduling
- Sporadic scheduling

With FIFO scheduling, a thread continues to run until it voluntarily relinquishes control or is preempted by a higher-priority thread. With Cisco IOS XR, only the idle threads of procnto (kernel) use FIFO scheduling.

Most other processes in IOS XR use round-robin scheduling, which restricts the maximum amount of time (timeslice) a thread can run without relinquishing control. A thread that uses round-robin scheduling runs until it voluntarily relinquishes control, gets preempted by a higher-priority thread, or consumes its timeslice.

Sporadic scheduling algorithm allows a thread to run at its normal priority for a certain amount of time (budget) over a period of replenishment interval before its priority is dropped to a lower priority. Figure 2-3 illustrates how sporadic scheduling works.

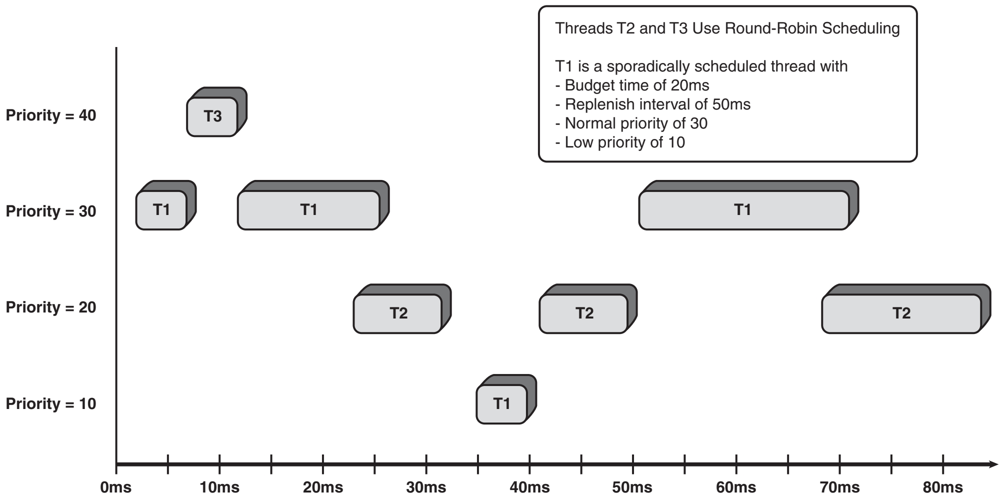


Figure 2-3 *Sporadic Scheduling*

Assume that at time t = 0 ms, threads T1 and T2 are ready to run and all other threads are blocked. Furthermore, assume that T1 is a sporadically scheduled thread with normal priority of 30, low priority of 10, budget time of 20 ms, and replenish interval of 50 ms. T2 is a round-robin scheduled thread with a priority of 20. An example follows:

1. T1 is scheduled to run at t = 0 ms because it has a higher priority than T2.
2. At t = 5 ms, thread T3 with priority of 40 is unblocked and becomes ready.

3. Because T3 has a higher priority than T1, it preempts T1 and starts running. As a result, T1 is moved to the head of the ready queue for priority 30.
4. At t = 10 ms, T3 is blocked and relinquishes CPU. Note that threads T1 with priority 30 and T2 with priority 20 are in the ready state. Therefore, because of its high priority, T1 starts running at t = 10 ms.
5. At t = 25 ms, because T1 has already used its budget time of 20 ms and is a sporadically scheduled thread, its priority is reduced to 10. Because T2 is in ready state and has a priority of 20, it preempts T1 and starts running at t = 25 ms.
6. At t = 50 ms, after the replenish interval lapses, the priority of T1 is restored to its normal priority of 30, which causes T1 to preempt T2 and start running.
7. At t = 70 ms, after T1 has used its budget in the new replenish interval, its priority is reduced to 10 again. This causes T2 to preempt T1 and start running.

Example 2-2 shows a partial output of **show process pidin**, which lists the threads and corresponding process ID (pid), thread ID (tid), process name, priority, scheduling algorithm, and state. The prio column shows the priority and scheduling algorithm of the thread. The scheduling of a thread is denoted as f for FIFO, r for round-robin, or ? for sporadic scheduling. Example 2-2 shows that there are two threads that use FIFO scheduling: procnto threads 1 and 2. These are the idle threads of the kernel. There is one idle thread per CPU. Because this output was taken from the RP of CRS-16/s, which has a two-processor CPU complex, it shows two idle threads, one for each CPU. Example 2-2 also shows that threads 3, 4, and 7 of the eth_server process use sporadic scheduling. All other threads use round-robin scheduling.

Example 2-2 *Output of* show processes pidin

```
RP/0/RP0/CPU0:CRS-A#show processes pidin
     pid tid name               prio STATE       Blocked
       1   1 procnto              0f READY
       1   2 procnto              0f RUNNING
       1   3 procnto             63r RECEIVE     1
       1   4 procnto             10r NANOSLEEP
       1   5 procnto             63r RECEIVE     1
       1   6 procnto             10r RECEIVE     1
       1   7 procnto             63r RECEIVE     1
       1   8 procnto             63r RECEIVE     1
       1   9 procnto             63r RECEIVE     1
...
   40987   3 pkg/bin/eth_server  50? SEM         29df078
   40987   4 pkg/bin/eth_server  49? SEM         29df080
   40987   5 pkg/bin/eth_server  10r SEM         29defc8
   40987   6 pkg/bin/eth_server  10r RECEIVE     5
   40987   7 pkg/bin/eth_server  55? RECEIVE     9
   40987   8 pkg/bin/eth_server  10r RECEIVE     12
   40987   9 pkg/bin/eth_server  10r RECEIVE     1
```

```
    40987  10 pkg/bin/eth_server  10r RECEIVE     1
    40987  11 pkg/bin/eth_server  55r RECEIVE     1
    45084   1 kg/bin/bcm_process  10r RECEIVE     1
    45084   2 kg/bin/bcm_process  56r RECEIVE     6
    45084   3 kg/bin/bcm_process  56r INTR
    45084   4 kg/bin/bcm_process  56r RECEIVE     10
    45084   5 kg/bin/bcm_process  56r RECEIVE     9
    45084   6 kg/bin/bcm_process  56r RECEIVE     1
    45085   1 pkg/bin/attachd     10r RECEIVE     1
    45085   2 pkg/bin/attachd     55r REPLY       40987
    45085   3 pkg/bin/attachd     55r REPLY       16397
    45086   1 ad_eeprom_protocol  10r RECEIVE     1
    45087   2 pkg/bin/qnet        10r RECEIVE     1
    45087   3 pkg/bin/qnet        10r RECEIVE     4
...
RP/0/RP0/CPU0:CRS-A#

! Count threads that use round-robin scheduling
RP/0/RP0/CPU0:CRS-A#show processes pidin | utility egrep -e "[0-9]+r " count
1025
RP/0/RP0/CPU0:CRS-A#

! List threads that use sporadic scheduling
 RP/0/RP0/CPU0:CRS-A#show processes pidin | include "[0-9]+\\? "
    40987   3 pkg/bin/eth_server  50? SEM          29df078
    40987   4 pkg/bin/eth_server  49? SEM          29df080
    40987   7 pkg/bin/eth_server  55? RECEIVE     9
   180321   5 /bin/parser_server  16? CONDVAR     485f48b4
RP/0/RP0/CPU0:CRS-A#

! List threads that use FIFO scheduling
RP/0/RP0/CPU0:CRS-A#show processes pidin | include "[0-9]+f "
        1    1 procnto              0f READY
        1    2 procnto              0f RUNNING
RP/0/RP0/CPU0:CRS-A#RP/0/RP0/CPU0:CRS-A#
```

Synchronization Services

The microkernel provides a message-passing–based synchronous IPC mechanism. This message-passing service copies a message directly from the address space of the sender thread to the receiver thread without intermediate buffering. The content and format of the message are transparent to the kernel. IPC is discussed in greater detail in the section “Interprocess Communication” later in this chapter.

In addition to the message-passing IPC mechanism provided by the microkernel, it is possible to develop other IPC mechanisms that use shared memory space. However, access to the shared memory space must be synchronized to ensure data consistency. For example, if one thread attempts to access a linked list while another thread is in the process of updating it, the result could be catastrophic. The microkernel provides mutex, condvar, and semaphore synchronization tools to address this problem.

Mutual exclusion lock, or mutex, is used to ensure exclusive access to data shared between threads. Before a thread can access the shared data it should first acquire (lock) the mutex. When it completes operation on the shared data, it releases the mutex. Only one thread may acquire a mutex at any given time. If a thread attempts to lock a mutex that is already locked by another thread, it will be blocked until the mutex is unlocked and acquired. When a thread releases a mutex, the highest priority thread waiting to acquire the mutex will unblock and become the new owner of the mutex.

If a higher-priority thread attempts to lock a mutex that is already locked by a lower-priority thread, the priority of the current owner will be increased to that of the higher-priority blocked thread. This is known as *priority inheritance* and solves the priority inversion problem. Priority inheritance and priority inversion are also discussed earlier in this section in the context of client/server thread interaction.

A conditional variable (condvar) is used to wait until some condition (for example, a timeout) is fulfilled. The thread blocks until the condition is satisfied. A condvar is usually used in conjunction with a mutex as follows:

- Lock a mutex
- Wait on a condvar
- Perform an activity (manipulate shared data)
- Unlock the mutex

Semaphore is another form of synchronization in which a thread waits for the semaphore to be positive. If the semaphore is positive, the thread unblocks and decrements the semaphore by 1. A post operation on a semaphore increments it by 1. A semaphore can be used to wake a thread by a signal handler. The thread issues a wait operation on the semaphore to wait for a signal. A signal handler will perform a post operation on the semaphore to wake a thread blocked by the semaphore.

As shown in Example 2-2 earlier in this section, **show process pidin location** *<r/s/m>* shows the state of each thread. Table 2-1 provides a list of states a process may take.

Table 2-1 *Process States*

State	Explanation
dead	The kernel is waiting to release the thread's resources.
running	Actively running on a CPU.
ready	Not running on a CPU but is ready to run.
stopped	Suspended (SIGSTOP signal).
send	Waiting for a client to send a message.

State	Explanation
receive	Waiting for a server to receive a message.
reply	Waiting for a server to reply to a message.
stack	Waiting for more stack to be allocated.
waitpage	Waiting for the process manager to resolve a page fault.
sigsuspend	Waiting for a signal.
sigwaitinfo	Waiting for a signal.
nanosleep	Sleeping for a period of time.
mutex	Waiting to acquire a mutex.
condvar	Waiting for a conditional variable to be signaled.
join	Waiting for the completion of another thread.
intr	Waiting for an interrupt.
sem	Waiting to acquire a semaphore.

As seen in Example 2-2, some of the threads are in a Reply state, which indicates that the client thread is blocked waiting for a reply. If a process is stuck in blocked state, it might be an indication of a problem with the (client/server) process or application. However, the existence of blocked processes on the router does not necessarily indicate a problem because it is expected behavior for selected processes. Other processes stuck in a block state might cause applications not to respond. It is important to understand the typical behavior of a router in your production network.

To display a list of blocked threads, issue the command **show processes blocked location** <*r/s/m*>, as shown in Example 2-3. It is recommended to issue this command several times within a few seconds. Running the command numerous times verifies whether processes are questionably blocked in error versus in a blocked state as it performs its normal IPC exchange. Processes such as ksh and devc-conaux are in a blocked state by design. Ksh is the client process communicating to devc-conaux (the server process). Here the thread is in blocked state until a user provides input on the console server. More specifically, ksh waits for input on the console or the auxiliary port and returns a system message to devc-conaux. When devc-conaux replies to ksh, the process changes the state from blocked to reply.

Example 2-3 *Processes Block*

```
RP/0/RP0/CPU0:CRS1-4#show processes blocked location 0/rp0/cpu0
  Jid       Pid Tid            Name State   TimeInState    Blocked-on
65546     12298   1             ksh Reply  101:26:48:0708   12296  devc-conaux
   52     40988   2         attachd Reply  101:26:50:0679   40985  eth_server
   52     40988   3         attachd Reply  101:26:50:0678   16397  mqueue
   78     40990   6            qnet Reply    0:00:00:0040   40985  eth_server
   78     40990   7            qnet Reply    0:00:00:0038   40985  eth_server
   78      40990   8             qnet Reply    0:00:00:0032   40985  eth_server
   78      40990   9             qnet Reply    0:00:00:0041   40985  eth_server
   78      40990  10             qnet Reply    0:00:00:0028   40985  eth_server
   78      40990  11             qnet Reply    0:00:00:0033   40985  eth_server
   78      40990  12             qnet Reply    0:00:00:0033   40985  eth_server
   78      40990  13             qnet Reply    0:00:00:0039   40985  eth_server
   51      40996   2    attach_server Reply  101:26:50:0438   16397  mqueue
  394     172114   1      tftp_server Reply  101:25:02:0994   16397  mqueue
  135     499850   4           cethha Reply    0:00:10:0981       1  node
0/RP1/CPU0 kernel
  276     512195   2          lpts_fm Reply    0:00:10:0194  495725  lpts_pa
65742    1470670   1             exec Reply   42:18:14:0620   12296  devc-conaux
65807    1474831   1             exec Reply    2:14:28:0875  512187  devc-vty
65809    4870417   1             exec Reply    0:00:00:0300       1  kernel
65810    4874514   1             more Reply    0:00:00:0094   16395  pipe
65811    4874515   1   show_processes Reply    0:00:00:0000       1  kernel
RP/0/RP0/CPU0:CRS1-4#
```

Cisco IOS XR System Manager

Cisco IOS XR has hundreds of processes running simultaneously on multiple nodes. Some processes are associated with applications and protocols. Examples of such processes include telnetd and isis, which refer to Telnet daemon (server) and the IS-IS routing protocol, respectively. Other processes are dedicated to system functions such as device drivers, interprocess communication, system health monitor, file system, configuration management, software install management, and so on. Such processes are always operational.

IOS XR system manager is the central entity responsible for starting, monitoring, restarting, terminating, and core dumping most IOS XR processes during bootup, RP failover, software activation, and in response to router configuration. System manager can also initiate disaster recovery based on process health. System manger runs on each route processor and line card in the system. Two instances of sysmgr process are running on each node. One of the instances is the primary sysmgr, which is responsible for all system manager responsibilities; the second instance acts as a standby and is ready to assume the primary role if the current primary sysmgr exits for some reason.

The following are the main functions of system manager:

- Start processes during bootup or node reload
- Start processes during route processor (RP) failover
- Start processes in response to user configuration; for example, when a user configures an OSPF process using **router ospf** <*process-name*>, system manager starts a new OSPF process instance

- Act as a central repository for all process-related information
- Initiate disaster recovery based on the process health
- Invoke dumper to collect a core dump when a process terminates abnormally

Process Attributes

System manager uses process attributes stored in a startup file for each process that it manages. Each startup file corresponds to an executable. The startup files are located in the /pkg/startup/ directory and contain tokens that are used by system manager to manage the corresponding process. The startup file for OSPF, FIB manager, and GSP are shown in Example 2-4.

Example 2-4 *Startup Files for OSPF, FIB_mgr, and GSP Processes*

```
# more /pkg/startup/ospf.startup
name:ospf
path:/ios/bin
item:/cfg/gl/ipv4-ospf/proc/
copies:10
tuple_dynamic_tag:ON
placement: ON
check_avail: ON
failover_tier: isis
standby_capable: ON
#
# more /pkg/startup/fib_mgr.startup
name:fib_mgr
path:/ios/bin
check_avail: ON
level:99
standby_capable:ON
#
# more /pkg/startup/gsp-rp.startup
name:gsp
path:/ios/bin
level: 80
check_avail:on
mandatory: on
```

The name token corresponds to the name of the process and the corresponding executable. The path token is the path where the executable is located. If the level token is set it implies that the process is level started during boot. During system boot the system manager uses the level token to determine the sequence in which the processes are started. The startup sequence during boot is shown in Table 2-2. You can use **show process boot location** <*r/s/m*> to see the boot sequence and startup level and at what time each process is ready.

Table 2-2 *IOS XR Startup Levels During Boot*

Band	Levels	Examples
MBI	0–39	dllmgr, nvram, obflmgr, dumpr, syslogd
ARB (arbitration)	40	wdsysmon, redcon
ADMIN	41–90	sysdb, netio, oir_daemon, gsp, envmon, shelfmgr
INFRA	91–100	ifmgr, aib, ipv4_io, fib_mgr
ACTIVE	101–150	ipv4_ma
FINAL	151–999	clns, arp

If the item token is set, it indicates that the corresponding process is started or terminated when a user enters configuration or when the configuration is loaded to the system database (sysdb) during boot. The process is started when the configuration item specified in the startup file is added to sysdb. For example, when the user enters and commits the configuration **router ospf** <*process-name*> system manager starts an OSPF process instance. System database is discussed in more detail in the section "Cisco IOS XR System Database" later in this chapter.

If the mandatory token is set to ON, the process is considered critical for the functioning of the node, which is any subsystem such as line card, route processor, service processor, or switch fabric card module running IOS XR. If a mandatory process exits or dies and fails to be restarted after repeated attempts, system manager will reload the node.

If the placement item is set to ON, the corresponding process is placeable and can run on any active RP or DRP node on the router. Process placement is discussed in the section "Process Placement," later in this chapter.

System manager assigns a unique job id (JID) to each executable. The JID is persistent across restarts. In addition to a JID each process is assigned a unique process ID (PID) when it is started or restarted. If a process is restarted it is assigned a new PID number but it retains its original JID number.

Some processes, such as telnetd, are transient processes that are started in response to a user request (telnetting to the router). These transient processes are not managed by system manager, and their JID is derived from their processed ID (PID). For transient processes a startup file is not needed because they are not started by the system manager.

System Manager and Process Lifecycle

System manager monitors the health of each process. Figure 2-4 shows the system manager and process lifecycle. When system manager starts a process it starts an end of initialization (EOI) timer. After the process starts and completes initialization it notifies system manager by sending an end of initialization signal. If the EOI timer for a process expires before sysmgr receives EOI from the process, sysmgr declares that process initialization failed.

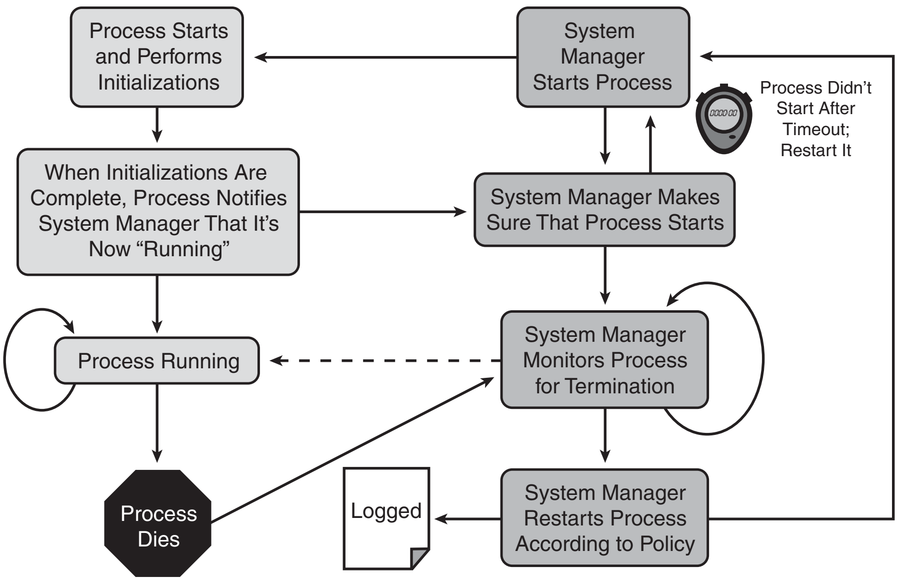

Process Starts and Performs Initializations
System Manager Starts Process
Process Didn't Start After Timeout; Restart It
When Initializations Are Complete, Process Notifies System Manager That It's Now "Running"
System Manager Makes Sure That Process Starts
Process Running
System Manager Monitors Process for Termination
Process Dies
Logged
System Manager Restarts Process According to Policy

Figure 2-4 *Sysmgr and Process Lifecycle*

CLI for Sysmgr and Processes

System manager provides a rich set of commands to check the status of processes; start, stop, crash, and restart processes; and configure process attributes. Use **show process** [<*process-name*> | <*JID*>] **location** <*r/s/m*> to show the process data and status. Example 2-5 displays an example of such output for an IS-IS process on a primary RP.

Example 2-5 *IS-IS Process and Threads*

```
RP/0/RP0/CPU0:CRS1-4#show processes isis
                  Job Id: 255
                     PID: 12714252
         Executable path: /disk0/hfr-rout-3.6.2/bin/isis
              Instance #: 1
              Version ID: 00.00.0000
                 Respawn: ON
           Respawn count: 5
   Max. spawns per minute: 12
            Last started: Thu Jan 15 05:06:57 2009
           Process state: Run (last exit due to SIGTERM)
           Package state: Normal
       Started on config: cfg/gl/isis/instance/test/ord_A/running
                    core: COPY
```

```
                Max. core: 0
                Placement: ON
             startup_path: /pkg/startup/isis.startup
                    Ready: 2.615s
                Available: 2.664s
         Process cpu time: 0.537 user, 0.130 kernel, 0.667 total
JID     TID  Stack pri state         TimeInState       HR:MM:SS:MSEC NAME
255     1      48K  10 Receive         0:00:14:0377    0:00:00:0284 isis
255     2      48K  10 Receive         0:00:01:0591    0:00:00:0075 isis
255     3      48K  10 Receive       110:34:27:0344    0:00:00:0008 isis
255     4      48K  10 Receive       110:34:18:0331    0:00:00:0002 isis
255     5      48K  10 Receive         0:00:01:0591    0:00:00:0159 isis
255     6      48K  10 Receive         0:00:07:0283    0:00:00:0009 isis
-------------------------------------------------------------------------------
RP/0/RP0/CPU0:CRS1-4#
```

The definition of the key attributes of IOS XR processes follow:

- **Job ID (JID):** The JID number remains constant, including process restarts.
- **Process ID (PID):** The PID field changes when a process is restarted.
- **Thread ID (TID):** A single process can have multiple threads executing specific tasks for a process.
- **Executable path:** References a path to the process executable. An additional field called "executable path on reboot" may appear if an in-service software upgrade has been performed.
- **Instance:** There may be more than one instance of a process running at a given time. Each instance is referenced by a number.
- **Respawn count:** The number of times a process has been (re)started. The first time a process is started, the respawn count is set to 1. Respawn mode is on or off. This field indicates whether this process restarts automatically in case of failure.
- **Max spawns per minute:** The number of respawns not to be exceeded in one minute. If this number is exceeded, the process stops restarting as a self-defense mechanism.
- **Last started date and time:** This timestamp shows when the process was last started.
- **Process state:** This shows the current state of the process.
- **Started on config:** Points to the location in system database (SysDB) that contains configuration data that resulted in spawn of the process.
- **core:** Memory segments to include in core file.
- **Max. core:** Shows the number of times to dump a core file. A value of 0 signifies infinity.