Solaris Resource Manager 1.3 System Administration Guide

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 816–7751–10
December 2002

020930@4660

Contents

Figures

Preface

This guide is for system administrators who are responsible for configuring and administering Solaris™ Resource Manager 1.3 software in Solaris 2.6 *SPARC™ Platform Edition*, Solaris 7 *SPARC Platform Edition*, and Solaris 8 *SPARC Platform Edition*. This document is applicable to all releases of the Solaris Resource Manager product.

Before You Read This Book

Before using this book, you should be familiar with the information in the Solaris 8 System Administrator Collection, available at http://docs.sun.com.

How This Book Is Organized

This book is organized into chapters, an appendix, and a glossary.

Chapter 1 provides an introduction to Solaris Resource Manager and describes how this product can be used to allocate and control major system resources.

Chapter 2 discusses the operation of the Solaris Resource Manager software and illustrates a simple hierarchy.

Chapter 3 describes how to configure the Solaris Resource Manager software on your Solaris system.

Chapter 4 describes the effects of the Solaris boot procedure on the Solaris Resource Manager product.

Chapter 5 discusses the per-user structure introduced in Solaris Resource Manager.

Chapter 6 discusses the scheduler, which is used to control the allocation of the CPU resource.

Chapter 7 describes how to control the amount of virtual memory held by users and individual processes.

Chapter 8 discusses how to regulate the resource consumption of physical memory by collections of processes. This feature is only available in the Solaris 8 operating environment.

Chapter 9 describes the mechanism for collecting accrued usage values for CPU, application, and user resources.

Chapter 10 describes batch process control, databases, web server management, and processor sets in detail and provides usage examples. It also describes configuration in a Sun™ Cluster 3.0 update environment.

Chapter 11 provides assistance in diagnosing problems in the operation of Solaris Resource Manager.

Chapter 12 describes possible error messages and their meanings.

Appendix A provides sample scripts.

Glossary is a list of words and phrases found in this book and their definitions.

Related Books

The following resources provide installation, configuration, usage, and release information for the Solaris Resource Manager product:

- The*Solaris Resource Manager 1.3 Release Notes* document provides a brief introduction to the Solaris Resource Manager product and contains descriptions of bugs and known problems.
- The *Solaris Resource Manager 1.3 System Installation Guide* describes how to install the Solaris Resource Manager software on your operating system. It is included in the product box.
- The *Solaris Resource Manager 1.3 Reference Manual* is the AnswerBook2 version of the Solaris Resource Manager man pages. These entries supplement the Solaris base manual pages installed on your system. This administration guide references these pages. Online versions of these man pages, accessible using the `man` command, are also provided in the Solaris Resource Manager `SUNWsrm` and `SUNWrcapm` packages.

The books listed above are also located at http://docs.sun.com.

For additional information on Sun Cluster, see the Sun Cluster 3.0 12/01 Collection available at `http://docs.sun.com`.

Note – If you are using a Sun Cluster 3.0 update release later than 12/01, refer to the appropriate Sun Cluster documentation for the release you are using.

Accessing Sun Documentation Online

The docs.sun.com[SM] Web site enables you to access Sun technical documentation online. You can browse the docs.sun.com archive or search for a specific book title or subject. The URL is `http://docs.sun.com`.

Typographic Conventions

The following table describes the typographic changes used in this book.

TABLE P–1 Typographic Conventions

Typeface or Symbol	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories; on-screen computer output	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% you have mail.`
`AaBbCc123`	What you type, contrasted with on-screen computer output	`machine_name%` **`su`** `Password:`
AaBbCc123	Command-line placeholder: replace with a real name or value	To delete a file, type **`rm`** *filename*.
AaBbCc123	Book titles, new words, or terms, or words to be emphasized.	Read Chapter 6 in *User's Guide*. These are called *class* options. You must be *root* to do this.

Shell Prompts in Command Examples

The following table shows the default system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

TABLE P–2 Shell Prompts

Shell	Prompt
C shell prompt	`machine_name%`
C shell superuser prompt	`machine_name#`
Bourne shell and Korn shell prompt	`$`
Bourne shell and Korn shell superuser prompt	`#`

CHAPTER 1

Overview

The Solaris Resource Manager software ensures resource availability for users, groups, and applications. It provides the ability to allocate and control major system resources such as CPU, virtual memory, physical memory (available on Solaris Resource Manager 1.3 on Solaris 8 only), and number of processes. It also implements administrative policies that govern which resources different users can access, and more specifically, how much of a particular resource each user is permitted to use. The Solaris Resource Manager product is a key enabler for server consolidation and increased resource utilization.

Introduction to Solaris Resource Manager

Organizational Goals It Addresses

IT organizations are often required to control costs and guarantee service levels for enterprise applications. Resource management can make it possible to lower overall total cost of ownership, have more accurate control over who uses the system and how they use it, and sometimes serve both goals.

By using Solaris Resource Manager software to categorize and prioritize usage, administrators can effectively utilize reserve capacity during off-peak periods, often eliminating the need for additional processing power.

By segregating workloads within the system, Solaris Resource Manager enables the system administrator to run and manage dissimilar applications on a single system, rather than dedicating an entire system—complete with peak capacity—to each

application. Traditionally, the most common approach to ensuring predictable service and response time is to host one function per system. This method works, but the proliferation of systems in the data center is expensive and difficult to manage.

Solaris Resource Manager allows consolidation of several applications on a single UNIX® server, thus fully utilizing all available resources. At the same time, all users can receive resources commensurate with their service levels and the relative importance of their work.

Key Solaris Resource Manager Features

The following table identifies and briefly describes key system resource control features discussed in this manual.

System Resource	Solaris Resource Manager Parameter	Description
CPU shares	`cpu.shares`	The amount of CPU time allocated to an lnode, specified in the limits database file as a number of shares. Solaris Resource Manager allocates all available system resources; that is, an lnode can receive more than its allocation when the resources are available.
CPU accrual	`cpu.accrue`	The accrued CPU usages for all lnodes within the group as well as that of the current lnode.
Memory limit	`memory.limit`	The maximum virtual memory usage allowed for all processes attached to an lnode. This limit is a fixed value specified in the limits database file. A value of zero indicates that no limit is set.
Process memory limit	`memory.plimit`	The maximum per-process virtual memory limit. This is a fixed value specified in the limits database file. A value of zero indicates that no limit is set.
Physical memory limit	`rss.limit` (Solaris 8 only)	The physical memory cap, in specified units, that is applied to collections of processes attached to an lnode. The cap must be a positive number.
Memory accrual	`memory.accrue`	The overall memory resources used over a period of time. Value is measured in byte-seconds.

System Resource	Solaris Resource Manager Parameter	Description
Number of processes	`process.limit`	Limits the number of processes that a user can run simultaneously, according to fixed limits specified in the limits database file.
Number of logins per user	`flag.onelogin` `flag.nologin`	Prohibits login or limits the number of simultaneous login sessions per user and/or scheduling group in accordance with fixed limits specified in the limits database file. Solaris Resource Manager tracks login count using PAM authentication records and `utmp`(4) entries. The counter is incremented and decremented dynamically.

Connect-time	`terminal.limit`
	`terminal.usage`
	`terminal.accrue`
		Solaris Resource Manager dynamically tracks a user's connect-time and compares it to the fixed limit specified in the limits database file by the system administrator or group header. As a user approaches the connect-time limit, Solaris Resource Manager sends warning messages to the user's terminal. When the limit is reached, the user is notified and then logged out forcibly after a short grace period.

When to Use Solaris Resource Manager

Solaris Resource Manager can provide effective resource control in a variety of situations including server consolidation, Internet Service Provider (ISP) web hosting, batch processing, administering sites with large or varied user populations, and establishing policies to ensure that critical applications get the response time they require.

Server Consolidation

Solaris Resource Manager is ideal for environments that are consolidating multiple applications on a single server. The cost and complexity of managing numerous machines encourages system managers to consolidate applications on larger, more scalable systems. With Solaris Resource Manager, it's easy to achieve these economies of scale.

As an example, a single Sun server could provide application, file, and print services for heterogeneous clients, messaging/mail service, web service, and mission-critical database applications. Since Sun Enterprise™ servers scale from 1 to 64 processors, one server could be configured for several departments to share or for an entire enterprise to use. In other server consolidation efforts, the development, prototype, and production environments are combined on a single large machine such as the Sun

Enterprise 10000 or Sun Enterprise 6500, rather than being hosted on three separate servers. Still other consolidation projects combine database and application servers within a single machine, or multiple data marts. Regardless of the application type or configuration, Solaris Resource Manager helps ensure that the system's resources are allocated among all users, applications, and groups according to the defined policy. Critical applications are protected because they are guaranteed the share of the available system resources they need.

Web Hosting

In the past, ISPs have had to assign dedicated machines to each client, at significant cost and complexity. With Solaris Resource Manager, an ISP can confidently host many (perhaps thousands) of web servers on a single machine. Solaris Resource Manager allows administrators to control the resource consumption associated with each web site, protecting each from the potential excesses of the others. It also prevents a faulty common gateway interface (CGI) script from exhausting CPU resources, or a user application from leaking all available virtual memory.

Batch Processing

Solaris Resource Manager can be used to prevent batch workloads from impacting ongoing business activities as well as other batch jobs running concurrently.

Large or Varied User Population

Solaris Resource Manager can help manage resources in any system that has a large and diverse user base, such as an educational institution. (In fact, Solaris Resource Manager has its roots in an early CPU resource scheduler developed at the Universities of Sydney and New South Wales.) Where there is a mix of workloads, the Solaris Resource Manager software can be configured to favor certain users. For example, in large brokerage firms, traders intermittently require fast access to execute a query or perform a calculation. Other system users, however, have more consistent workloads. If the traders are granted a proportionately larger amount of processing power, Solaris Resource Manager ensures that they will have the responsiveness they need.

Main Capabilities

Solaris Resource Manager provides the ability to administer the consumption of various important resources in the system, such as processor time, virtual memory, process count, login control, and connect-time. The Solaris Resource Manager administrative model adds flexibility by permitting the delegation of administrative

rights within a hierarchy, relieving the data center staff from the need to be involved in intra-group administrative transactions. In addition, Solaris Resource Manager provides mechanisms for collecting resource usage data that can be applied to capacity planning or chargeback purposes.

Processing Resources

One of the fundamental jobs of the operating system is to arbitrate which processes get access to the system's resources. The default Solaris timeshare (TS) scheduler tries to give every process relatively equal access to the system's resources. Limitations on access are applied to processes without physical memory resources, which are not permitted to run, and processes with pending I/O requests, which are blocked.

This scheme is the basis for most modern operating systems; it works well as long as "equal access for all" is a suitable policy for the organization. However, more sophisticated mechanisms are needed to implement different policies. For example, a manufacturing department might own a large system that is usually used lightly because of fluctuating seasonal demand. At the same time, the engineering department almost always needs more computational cycles. Although it is wasteful to underuse a large machine's resources, sharing the manufacturing system with engineering has traditionally been problematic. With simple scheduling policies, there is no way to express to the operating system that the manufacturing department's users are more important than the engineering users on the same system. If manufacturing has a critical job running that consumes 75 percent of the system's resources, the job will make suitable progress if all other jobs request 25 percent of the system or less. However, if an engineering job arrives that demands 50 percent of the system, that critical manufacturing job will likely not get what it needs to maintain adequate headway because the system will try to accommodate both jobs on an equal basis.

Now assume that the administrator determines that manufacturing's normal processing requirements can be met with 80 percent of the machine's capabilities. Using Solaris Resource Manager, the system administrator can specify that the manufacturing department's users can have up to 85 percent of the system's processing capability if they request it, and the scheduler will apportion the remainder to any other user. A more extreme but equally valid configuration might specify that manufacturing users can have up to 100 percent of the system if necessary, effectively preventing any other group's processes from running in the event that manufacturing really needs the entire system.

Allocating Resources

Solaris Resource Manager has a CPU scheduling class that replaces the standard timesharing scheduler. Called the SHR scheduler, this module implements what is called a fair share scheduler. The term is something of a misnomer, because it is the

system administrator who specifies what "fair" means. In the example above, "fair" meant that manufacturing could get 100 percent of the system. The SHR scheduler is responsible for allocating resources according to the plan laid out in the administrative profile.

Restricting Resources

Solaris Resource Manager maintains a database of resource usage and associated limits.

The SHR scheduler takes into account the administrative specification for resource guarantees. It is capable of managing resources that are renewable (such as CPU time) or fixed (such as number of processes).

Other Solaris Resource Manager modules implement restrictions on the consumption of various resources. For example, connect-time and user logins are managed by a Pluggable Authentication Module (PAM). The PAM module consults the Solaris Resource Manager database each time a user attempts to log in. Once the system authenticates the user (generally through password matching), the user's connect-time and number of current logins are checked against the limits. The login is rejected if either limit is exceeded.

Relationship to Other Solaris Resource Control Features

The Solaris operating environment includes several other features that provide control over certain kinds of resources. Some features, such as real-time scheduling, `nice`(1), quotas, and processor sets, are part of the basic Solaris system.

Solaris Bandwidth Manager is a co-packaged software package, dynamic system domains are features of the Sun Enterprise 10000 system platform, and dynamic reconfiguration is a feature of the Sun Enterprise system platform.

All of these components offer types of resource management, but each differs from Solaris Resource Manager capabilities in some way.

- Real-time Scheduling

  The standard Solaris operating system uses the TS scheduling class for most conventional work, but it also offers real-time (RT) scheduling to users with sufficient privilege. The RT scheduling class implements a very different (and intentionally very weighted) scheduling policy to ensure that specific workloads or processes get immediate access to the processor.

  Solaris Resource Manager can coexist on the same system as the RT scheduling class, but it will have no control over any process running in the RT class. The Solaris Resource Manager fair share scheduler is able to manage the CPU time resources of only those processes that are not running in the RT scheduling class.

For example, on a four-processor system, a single-threaded process can consume one entire processor; in fact, this is precisely what happens if the requesting process is CPU-bound. If this system also runs Solaris Resource Manager, regular user processes will be competing for the three CPUs not already being used by the real-time process. (Note that the real-time process might not use the CPU continually. When it is idle, Solaris Resource Manager will utilize all four processors.)

- `nice`(1) Command

  The `nice` command permits a user to manipulate program execution priority. Unless superuser privilege is invoked, this command only permits the user to lower the priority. This can be a useful feature (for example, when a user starts a low-priority batch job from an interactive login session), but it relies on the cooperation of the user. Solaris Resource Manager enforces administrative policies, even without the cooperation of the user.

- Quotas

  Solaris file systems have quota mechanisms that enable the administrator to restrict the disk consumption of individual users. This functionality is independent of Solaris Resource Manager.

- Processor Sets

  Processor sets were introduced in Solaris 2.6. This feature permits the administrator to divide multiprocessor systems into logical groups and permits users to launch processes into those groups. The advantage is that workloads running in one processor set are protected from CPU activity taking place in any other processor set. In some ways, this is similar to what Solaris Resource Manager does, but the two features operate on a completely different basis. Processor sets control only CPU activity. The control is at a relatively broad hardware level, because processors can belong to only one processor set at a time. Especially in the case of relatively small systems, the granularity may be quite high: on a 4-processor system, the minimum resource that can be assigned is 25 percent of the system.

  Solaris Resource Manager has much finer-grained control; each user is allocated a share of the system. The shares can be distributed arbitrarily on a fine granularity, and the scheduler will allocate resources accordingly. For example, if 50 shares are granted and one user has 40 of them, that user will get 40 / 50 = 80 percent of the resource. Similarly, if 67 total shares are granted, a user with 57 shares will get 85 percent of the resource. In addition, Solaris Resource Manager can control resources other than CPU. See “The Role and Effect of Processor Sets” on page 96 for more information on the interaction of Solaris Resource Manager and processor sets.

- Dynamic System Domains

  The Sun Enterprise 10000 has a feature called dynamic system domains, which permit the administrator to logically divide a single system rack into one or more independent systems, much like the partitioning features available on mainframes. Each system runs its own copy of Solaris. For example, a machine with 32 CPUs on 8 system boards might be operated as 1 system with 16 CPUs, and 2 other systems

with 8 CPUs each. In this configuration, three copies of Solaris would be running. Dynamic system domains also provide tools that manage the transfer of resources into and out of each copy of Solaris, thus creating a relatively broad facility for managing physical resources. (The minimum unit of inter-domain allocation is an entire system board.)

Solaris Resource Manager is similar to dynamic system domains in that it also provides the administrator with mechanisms to allocate resources, but it does so in very different ways. Solaris Resource Manager runs within a single instance of Solaris, and provides a finer degree of administrative control over the resources in that system. Solaris Resource Manager can be used to segment resources among many users and applications in each instance of Solaris within a Sun Enterprise 10000 system and used in conjunction with dynamic system domains.

- Dynamic Reconfiguration

  The dynamic reconfiguration feature of Sun Enterprise servers enables users to dynamically add and delete system boards, which contain hardware resources such as processors, memory, and I/O devices. The effect of a dynamic reconfiguration operation on memory has no impact on Solaris Resource Manager memory-limit checking.

- Solaris Bandwidth Manager

  Solaris Bandwidth Manager is an unbundled package that works with the Solaris kernel to enforce limits on the consumption of network bandwidth. Solaris Bandwidth Manager is a form of resource management software that applies to a different class of resources. Solaris Resource Manager and Solaris Bandwidth Manager have different and separate management domains: Solaris Resource Manager operates on a per-user or per-application basis, while Solaris Bandwidth Manager manages on a per-port, per-service, or per-protocol basis.

Differences Between Solaris Resource Manager and Similar Products

The Solaris Resource Manager product is related to many other software components that might be present in the system, but it does not replace any of them.

- Solaris Resource Manager and Solaris Bandwidth Manager manage different types of resources.
- Solaris Resource Manager is not a system monitor in the sense that Sun Management Center is.
- Solaris Resource Manager is not really a capacity planning tool. While it helps the administrator manage capacity, and its accounting functions construct usage records that the administrative staff might use to do trend analysis, it does not do capacity planning in the traditional sense of the term.
- Solaris Resource Manager is not a job scheduler; it controls how a process runs on its host system, rather than when or where it runs.

- Because Solaris Resource Manager operates only on a single system, it is not a mechanism for implementing load balancing across cluster members. Solaris Resource Manager can be used effectively to manage workloads individually on each member of a cluster, however. For example, arrangements might be made to prioritize a workload from a failed member of a high availability cluster over a background workload running on standby member.

Solaris Resource Manager on Sun Cluster 3.0 Update Environments

The Sun Cluster software environment provides high availability (HA) support for data services and parallel database access on a cluster of servers. Sun Cluster prevents loss of service in the event of a single-point failure through hardware redundancy, hardware and software failure detection, failover of applications, and automatic restart of data services. Sun Cluster software includes a cluster framework and support for a set of HA agents. Sun Cluster also includes an application programming interface (API) that can be used to create HA applications and to integrate them with the Sun Cluster framework.

All Solaris Resource Manager features for controlling system resources are supported in the Sun Cluster environment. You can install Solaris Resource Manager on any valid Sun Cluster topology, on two-node or greater-than-two-node clusters. See "Configuring Solaris Resource Manager in Sun Cluster 3.0 Update Environments" on page 113 for configuration information.

For tips on installing Solaris Resource Manager in Sun Cluster 3.0 Update Environments, see the *Solaris Resource Manager 1.3 System Installation Guide*.

CHAPTER 2

Normal Operations

This chapter describes Solaris Resource Manager principles of operation and introduces key concepts. “Workload Configuration” on page 33 provides an example to reinforce the descriptions and to illustrate a simple hierarchy. (A more complex hierarchy example is presented in Chapter 10.)

Limit Node Overview

Solaris Resource Manager is built around a fundamental addition to the Solaris kernel called the *lnode* (limit node). Lnodes correspond to UNIX user IDs (UIDs) and can represent individual users, groups of users, applications, and special requirements. Lnodes are indexed by UID; they are used to record resource allocation policies and accrued resource usage data by processes at the user, group, and application levels.

Although an lnode is identified by UID, it is separate from the credentials that affect permissions. The credential structure determines whether a process can read, write, and modify a file. The lnode structure is used to track the resource limits and usage.

In certain cases, a user might want to use a different set of limits. This is accomplished by using `srmuser`(1SRM) to attach to a different lnode. Note that this change does not affect the credential structure, which is still associated with the original UID, and that the process still retains the same permissions.

Resource Management

Hierarchical Structure

The Solaris Resource Manager management model organizes lnodes into a hierarchical structure called the *scheduling tree*. The scheduling tree is organized by UID: each lnode references the UID of the lnode's *parent* in the tree.

Each sub-tree of the scheduling tree is called a *scheduling group*, and the user at the root of a scheduling group is the *group header*. (The `root` user is the group header of the entire scheduling tree.) Setting the flag `flag.admin` delegates the ability to manage resource policies within the scheduling group to the group header.

Lnodes are initially created by parsing the UID file. After Solaris Resource Manager has been installed, the lnode administration command (`limadm`(1MSRM)) is used to create additional lnodes and assign them to parents. The scheduling tree data is stored in a flat file database that can be modified as required using `limadm`.

Although a UID used by an lnode does not have to correspond to a system account, with an entry in the system password map, it is recommended that a system account be created for the UID of every lnode. In the case of a non-leaf lnode (one with subordinate lnodes below it in the hierarchy), it is possible for the account associated with the lnode to be purely administrative; no one ever logs in to it. However, it is equally possible that it is the lnode of a real user who does log in and run processes attached to this non-leaf lnode.

Note that Solaris Resource Manager scheduling groups and group headers have nothing to do with the system groups defined in the `/etc/group` database. Each lnode in the scheduling tree, including group headers, corresponds to a real system user with a unique UID.

Hierarchical Limits

If a hierarchical limit is assigned to a group header, it applies to the usage of that user plus the total usage of all members of the scheduling group. This allows limits to be placed on entire groups, as well as on individual members. Resources are allocated to the group header, who can allocate them to users or groups of users that belong to the same group.

Processes

Every process is attached to an lnode. The `init` process is always attached to the `root` lnode. When processes are created by the `fork`(2) system call, they are attached to the same lnode as their parent. Processes may be re-attached to any lnode using a Solaris Resource Manager system call, given sufficient privilege. Privileges are set by the *central system administrator* or by users with the correct administrative permissions enabled.

Resource Control

The Solaris Resource Manager software provides control of the following system resources: CPU (rate of processor) usage, virtual memory, physical memory (Solaris 8 only), number of processes, number of concurrent logins of a user and/or a scheduling group, and terminal connect-time.

TABLE 2–1 Solaris Resource Manager Functions

System Resource	Allocation Policy	Control	Measurement	Usage Data
CPU Usage	Yes (per user ID)	Yes	Yes (per user ID)	Yes
Virtual Memory	Yes (per-user, per-process)	Yes (per-user, per-process)	Yes (per-user, per-process)	Yes
Physical Memory (Solaris 8 Only)	Yes	Yes	Yes	Yes
No. of Processes	Yes	Yes	Yes	Yes
User/Scheduling Group Logins	Yes	Yes	Yes	Yes
Connect-Time	Yes	Yes	Yes	Yes

Solaris Resource Manager keeps track of the usage of each resource by each user. Users may be assigned hard limits on resource usage for all resources except CPU. A hard limit will cause resource consumption attempts to fail if the user allows the usage to reach the limit. Hard limits are directly enforced by either the *kernel* or the software that is responsible for managing the respective resource.

A limit value of zero indicates no limit. All limit attributes of the `root` lnode should be left set to zero.

Generally all system resources can be divided into one of two classes: *fixed* (or non-renewable) *resources* and *renewable resources*. However, in Solaris Resource Manager 1.3 used in the Solaris 8 environment, a third class is introduced: the *soft limit*. The way in which resident set size (RSS) soft limits are indirectly enforced by the resource cap enforcement daemon means that the usage can temporarily exceed the limit. See Chapter 8 for additional information.

Solaris Resource Manager manages fixed and renewable resources differently.

Fixed Resources	Fixed or non-renewable resources are those which are available in a finite quantity. Examples include virtual memory, number of processes, concurrent logins of a user and/or a scheduling group, and connect-time. A fixed resource can be consumed (allocated) and relinquished (deallocated), but no other entity can use the resource before the owner deallocates it. Solaris Resource Manager employs a usage and limit model to control the amount of fixed resources used. *Usage* is defined as the current resource being used, and *limit* is the maximum level of usage that Solaris Resource Manager permits.
Renewable Resources	Renewable resources are those which are in continuous supply, such as CPU time. Renewable resources can only be consumed, and, once consumed, cannot be reclaimed. At any one time, a renewable resource will have limited availability and if not used at that time, it will no longer be available. (An analogy is sunlight. There is only a certain amount arriving from the sun at any given instant, but more will surely be coming for the next few million years.) For this reason, renewable resources can be reassigned to other users without explicit reallocation to ensure no waste. Solaris Resource Manager employs a usage, limit, and decay rate to control a user's rate of consumption of a renewable resource. Usage is defined as the total resource used, with a limit set on the ratio of usages in comparison to other users in the group. *Decay rate* refers to the period by which historical usage is discounted. The next resource quantum, for example, clock tick, will be allocated to the active lnode with the lowest decayed total usage value in relation to its allocated share. The decayed usage value is a measure of the total usage over time less some portion of historical usage determined by a half-life decay model.

CPU Resource Management

The allocation of the renewable CPU resource is controlled using a fair share scheduler called the Solaris Resource Manager *SHR scheduler*.

Scheduler Methodology

Each lnode is assigned a number of CPU *shares*. The processes associated with each lnode are allocated CPU resources in proportion to the total number of outstanding *active* shares (active means that the lnode has running processes attached). Only active lnodes are considered for an allocation of the resource, because only they have active processes running and need CPU time.

As a process consumes CPU ticks, the CPU usage attribute of its lnode increases. The scheduler regularly adjusts the priorities of all processes to force the relative ratios of CPU usages to converge on the relative ratios of CPU shares for all active lnodes at their respective levels. In this way, users can expect to receive at least their entitlements of CPU service in the long run, regardless of the behavior of other users.

The scheduler is hierarchical because it also ensures that groups receive their group entitlements independently of the behavior of the members. Solaris Resource Manager SHR scheduler is a long-term scheduler; it ensures that all users and applications receive a fair share over the course of the scheduler term. This means that when a light user starts to request the CPU, that user will receive commensurately more resource than heavy users until their comparative usages are in line with their relative "fair" share allocation. The more you use over your entitlement now, the less you will receive in the future.

Additionally, Solaris Resource Manager has a decay period, set by the system administrator, that does not track past usage. The decay model is one of half-life decay, where 50 percent of the resource has been decayed away within one half-life. This ensures that steady, even users are not penalized by short-term, process-intensive users. The half-life decay period sets the responsiveness, or term, of the scheduler; the default value is 120 seconds. A long half-life favors even usage, typical of longer batch jobs, while a short half-life favors interactive users. Shorter values tend to provide more even response across the system, at the expense of slightly less accuracy in computing and maintaining system-wide resource allocation. Regardless of administrative settings, the scheduler tries to prevent resource starvation and ensure reasonable behavior, even in extreme situations.

Scheduler Advantages

The primary advantage of the Solaris Resource Manager SHR scheduler over the standard Solaris scheduler is that it schedules users or applications rather than individual processes. Every process associated with an lnode is subject to a set of

limits. For the simple case of one user running a single active process, this is the same as subjecting each process to the limits listed in the corresponding lnode. When more than one process is attached to an lnode, as when members of a group each run multiple processes, all of the processes are collectively subject to the listed limits. This means that users or applications cannot consume CPU at a greater rate than their entitlements allow, regardless of how many concurrent processes they run. The method for assigning entitlements as a number of shares is simple and understandable, and the effect of changing a user's shares is predictable.

Another advantage of the SHR scheduler is that while it manages the scheduling of individual threads (technically, in Solaris, the scheduled entity is a lightweight process (LWP)), it also apportions CPU resources between users.

These concepts are illustrated by the following equation:

$$new\_SRM\_priority = current\_SRM\_priority + \frac{CPU\_usage \times (\#\ of\ active\ processes)}{(\#\ of\ shares)^2}$$

The *new_SRM_priority* is then mapped to the system priority. The higher the Solaris Resource Manager priority, the lower the system priority, and vice versa. Every decay period, *CPU_usage* is reduced by half and incremented with the most current usage.

Each user also has a set of *flags*, which are boolean-like variables used to enable or disable selective system privileges, such as login. Flags can be set individually per user, or be inherited from a parent lnode.

A user's usages, limits, and flags can be read by any user, but they can be altered only by users with the correct administrative privileges.

Eliminating CPU Waste

Solaris Resource Manager never wastes CPU availability. No matter how low a user's allocation, that user is always given all the available CPU if there are no competing users. One consequence of this is that users may notice performance that is less smooth than usual. If a user with a very low *effective share* is running an interactive process without any competition, the job will appear to run quickly. However, as soon as another user with a greater effective share demands CPU time, it will be given to that user in preference to the first user, so the first user will notice a marked job slow-down. Nevertheless, Solaris Resource Manager goes to some lengths to ensure that legitimate users are not cut off and unable to do any work. All processes being scheduled by Solaris Resource Manager (except those with a maximum `nice` value) will be allocated CPU regularly by the scheduler. There is also logic to prevent a new user that has just logged in from being given an arithmetically "fair" but excessively large proportion of the CPU to the detriment of existing users.

Virtual Memory (Per-User and Per-Process Limits)

Virtual memory is managed using a fixed resource model. The virtual memory limit applies to the sum of the memory sizes of all processes attached to the lnode. In addition, there is a per-process virtual memory limit that restricts the total size of the process's virtual address space size, including all code, data, stack, file mappings, and shared libraries. Both limits are hierarchical. Limiting virtual memory is useful for avoiding virtual memory starvation. For example, Solaris Resource Manager will stop an application that is leaking memory from consuming unwarranted amounts of virtual memory to the detriment of all users. Instead, such a process only starves itself or, at worse, others in its resource group.

Physical Memory

If you are using Solaris Resource Manager 1.3 in the Solaris 8 operating environment, you can regulate the resource consumption of physical memory by collections of processes attached to an lnode or project. See Chapter 8 for information on this functionality.

Number of Processes

The number of processes that users can run simultaneously is controlled using a fixed resource model with hierarchical limits.

Terminal and Login Connect-Time Limits

The system administrator and group header can set terminal login privileges, number of logins, and connect-time limits, which are enforced hierarchically by Solaris Resource Manager. As a user approaches a connect-time limit, warning messages are sent to the user's terminal. When the limit is reached, the user is notified, then forcibly logged out after a short grace period.

Solaris Resource Manager progressively decays past usage of connect-time so that only the most recent usage is significant. The system administrator sets a half-life parameter that controls the rate of decay. A long half-life favors even usage, while a short half-life favors interactive users.

User Administration

The central system administrator (or superuser) can create and remove user lnodes. This administrator may alter the limits, usages, and flags of any user, including its own, and also set administrative privileges for any lnode, including assigning administrative privileges selectively to users.

A *sub-administrator* can be granted these privileges by the setting of the `flag.uselimadm` flag. A sub-administrator can execute any `limadm` command as user `root`, and can be thought of as an assistant to the superuser.

A user granted hierarchical administrative privilege by the setting of the `flag.admin` flag is called a *group administrator*. A group administrator can modify the lnodes of users within the sub-tree of which they are the group header, and manage the group's resource allocation and scheduling policy. Group administrators cannot alter their own limits or flags, and cannot circumvent their own flags or limits by altering flags or usages within their group.

Measurement

Resource usage information is visible to the administrators of the system, and provides two views of resource usage information: that based on each user, and a workload view of resource usage.

Usage Data Overview

The Solaris Resource Manager system maintains information (primarily current and accrued resource usage) that can be used by administrators to conduct comprehensive system resource accounting. No accounting programs are supplied as part of Solaris Resource Manager, but its utility programs provide a base for the development of a customized resource accounting system.

For more information on setting up accounting procedures, see Chapter 9.

Workload Configuration

The key to effective resource management using Solaris Resource Manager is a well-designed resource hierarchy. Solaris Resource Manager uses the lnode tree to implement the resource hierarchy.

Mapping the Workload to the Lnode Hierarchy

Each node in the lnode tree maps to a UID in the password map, which means that workloads must be mapped to align with entries in the password map. In some cases, additional users may need to be created to cater to the leaf nodes in the hierarchy. These special users do not actually run processes or jobs, but act as an administration point for the leaf node.

A Simple Flat Hierarchy

This simple hierarchy was constructed to control the processing resources of two users, Chuck and Mark. Both of these users consume large amounts of CPU resources at various points, and thus affect one other at different times during the day.

To resolve this, a single-level hierarchy is constructed, and equal shares of CPU are allocated to each user.

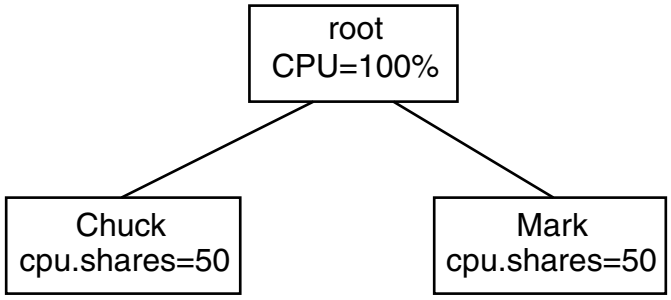


FIGURE 2–1 A Simple Flat Solaris Resource Manager Hierarchy

This simple hierarchy is established using the `limadm` command to make Chuck and Mark children of the `root` share group:

```
# limadm set sgroup=root chuck
# limadm set sgroup=root mark
```

To allocate 50 percent of the resources to each user, give each user the same number of CPU shares. (For simplicity, in this example 50 shares have been allocated to each user, but allocating 1 share to each would achieve the same result.) The `limadm` command is used to allocate the shares:

```
# limadm set cpu.shares=50 chuck
# limadm set cpu.shares=50 mark
```

Use the `liminfo` command to view the changes to the lnode associated with Chuck:

```
# liminfo -c chuck
Login name:                   chuck       Uid (Real,Eff):         2001 (-,-)
Sgroup (uid):              root (0)       Gid (Real,Eff):          200 (-,-)

Shares:                           50       Myshares:                   1
Share:                            41 %     E-share:                    0 %
Usage:                             0       Accrued usage:              0

Mem usage:                         0 B     Term usage:                 0s
Mem limit:                         0 B     Term accrue:                0s
Proc mem limit:                    0 B     Term limit:                 0s
Mem accrue:                        0 B.s

Processes:                         0       Current logins:             0
Process limit:                     0

Last used: Tue Oct 4 15:04:20 1998
Directory: /users/chuck
Name:       Hungry user
Shell:      /bin/csh

Flags:
```

The fields displayed by the `liminfo` command are explained in "A Typical Application Server" on page 109. Also refer to the `liminfo`(1SRM) man page for more information on `liminfo` fields.

CHAPTER 3

Configuration

Solaris Resource Manager provides a great deal of flexibility in its configuration to the central system administrator (for example, the `root` user). This chapter describes the following configuration areas:

- Kernel boot parameters used when the kernel is first started (see "Kernel Boot Parameters" on page 35).
- Global Solaris Resource Manager parameters supplied through the `srmadm`(1MSRM) command (see "Global Solaris Resource Manager Parameters via `srmadm`" on page 37).
- Parameters for the `limdaemon`(1MSRM) program (see "Using `limdaemon`" on page 40).
- PAM subsystem, including account and session management (see "PAM Subsystem" on page 41).

Kernel Boot Parameters

The kernel has certain Solaris Resource Manager parameters that can be set by the central administrator when the kernel is booted. The Solaris system reads the `/etc/system` file at boot time and uses it to configure kernel modules (see `system`(4) for details). The parameters that can be set in the SHR module (all are 32–bit integers) to override the Solaris Resource Manager default behavior are:

SRMLnodes The number of lnodes to cache in the kernel. On Solaris systems, each kernel lnode requires about 3 Kb. A value of zero (the default) means that the kernel will determine the value. The heuristic then used is:

```
(nproc / SRMProcsPerUid ) + SRMLnodesExtra
```

where *nproc* is the maximum number of simultaneous processes allowed in the system. A minimum value of `6` overrides this calculation. The maximum specified by *SRMMemoryMax* will also override this calculation.

SRMProcsPerUid The anticipated average number of processes used by each user. The default is 4.

SRMLnodesExtra A bias used in the heuristic to determine the size of the in-memory lnode array. The default is 20.

SRMNhash The number of entries in the hash table that is used to map UID values to lnodes in the kernel. On Solaris systems, each entry is 4 bytes long. The default is zero, which means to use the same value as for the number of lnodes.

SRMMemoryMax The reciprocal of this value is a fraction that specifies the maximum percentage of real memory to use for the Solaris Resource Manager lnode and hash tables combined. The default is 20, which means that a maximum of 5 percent of real memory will be used for Solaris Resource Manager data structures.

SRMMemWarnFreq The minimum interval, in seconds, between "memory exceeded" notification warnings for a single lnode. The default value is 4.

For example, in the `/etc/system` file the line:

```
set srmlim:SRMMemWarnFreq=10
```

will ensure that memory exceeded messages are not sent more frequently than once every 10 seconds for any single user.

There are also some parameters not in Solaris Resource Manager that affect its behavior. These include:

initclass This is the name of the scheduling class in which the `init`(1M) process is started. Under Solaris Resource Manager this should be given as the string `"SHR"` (including the double-quote characters). The default Solaris value is `"TS"`. To use Solaris Resource Manager for CPU resource control, the following line should be included in the `/etc/system` file:

```
set initclass="SHR"
```

to override the default.

extraclass This is a name of a scheduling class module to load, without necessarily using it as the default scheduling class. To use Solaris Resource Manager with only non-CPU resource control, the following line should be included in the `/etc/system` file:

```
set extraclass="SHR"
```

To boot a system that does not have Solaris Resource Manager loaded, an alternate `/etc/system` file named `/etc/system.noshrload` is used. See "Booting Without Solaris Resource Manager" on page 45 for instructions on this process.

Multi-User Startup Configuration

During a normal system boot, when the system changes from single-user to multi-user mode, a Solaris Resource Manager initialization script (see Appendix A) is run to set various Solaris Resource Manager parameters. Details of what this script does are given in Chapter 4.

If the initialization script itself (`/etc/init.d/init.srm`) is modified, copies of both the original and modified versions should be kept separately. Applying Solaris Resource Manager updates will not necessarily preserve existing initialization scripts.

Global Solaris Resource Manager Parameters via `srmadm`

The `srmadm` command allows an administrator to set, modify, or display the global Solaris Resource Manager parameters. Refer to the `srmadm`(1MSRM) man page for details of all parameters.

The `srmadm` command can be called any number of times to set various parameters. It is not necessary to include all settings on a single invocation. This also means that `srmadm` can be used to change the operational parameters of a running Solaris Resource Manager system on the fly, although some caution should be taken.

Of particular importance to administrators are the `srmadm` options that enable or disable the main features of Solaris Resource Manager. These are:

`fileopen[={y|n}]`

The default database is `/var/srm/srmDB`; it can be overridden with the `-f` option. Note that closing the Solaris Resource Manager database file in mid-operation should be regarded as an emergency action. It has several undesirable consequences: all processes will continue running on the surrogate `root` lnode, which may give them more privilege than normal; the SHR scheduler is disabled; and Solaris Resource Manager limit enforcement ceases. When disabled, Solaris Resource Manager has no limits database open, and its cache contains only the surrogate `root` lnode to which all processes are attached.

`share[={y|n}]`

When enabled, the Solaris Resource Manager SHR scheduler is used and CPU scheduling takes place according to the Solaris Resource Manager dynamic usage and decay algorithm. This mode cannot be set unless `fileopen` mode is enabled. When disabled, the SHR scheduler's usage calculations are frozen, and processes are scheduled "round-robin" with fixed equal priorities.

`limits[={y|n}]`

When enabled, Solaris Resource Manager enforces the virtual memory and process limits. This mode cannot be set unless `fileopen` mode is enabled. When disabled, Solaris Resource Manager will keep usage attributes up to date, but will not enforce limits.

`adjgroups[={y|n}]`

When enabled, the Solaris Resource Manager SHR scheduler's global group effective share adjustment is used. The enabled state is recommended in most circumstances. Every run interval, the normalized usages of all limits entries are recalculated. If the `adjgroups` scheduling mode is enabled, then extra processing of normalized usages is performed as follows. The scheduler makes a pass over the scheduling tree, comparing each group's recently received effective share with its entitlement. Groups that have received less than their group entitlement are biased to receive a greater effective share in the next run interval. This ensures that groups receive their entitlements of CPU service whenever possible, regardless of the actions of their members.

`limshare[={y|n}]`

When enabled, the SHR scheduler applies its priority ceiling feature to limit all users' effective shares, which prevents extremely low-usage users from briefly acquiring almost 100 percent of CPU. The enabled state is recommended.

The rate of CPU service for a user is roughly inversely proportional to the user's usage. If not active for a long time, a user's usage decays to near-zero. When such a user logs in (or the lnode becomes active in any way), then for the duration of the next run interval, the user's processes could have such high priority that they monopolize the CPU.

Enabling the `limshare` scheduling flag causes the scheduler to estimate the effective share that an lnode will receive before the next run interval. If the result exceeds the user's assigned entitlement by a given factor (see `maxushare`), the user's normalized usage is readjusted to prevent this.

There are two optional parameters to `srmadm` that are also useful to an administrator:

- The `-v` parameter prints a formatted report of all current parameter settings on standard output. Supplying more `-v` parameters (`-V 1`, `-V 2`, and `-V 3`) results in a more verbose report. Invoking `srmadm` with no arguments is equivalent to supplying a single `-v` option.
- The `-d` parameter initializes the Solaris Resource Manager system structure with default values instead of reading the current kernel settings. The default values, which mainly give control over scheduling behavior, are built into `srmadm` and provide a good starting point from which to customize Solaris Resource Manager. The kernel begins with the same values preset.

The following examples illustrate typical `srmadm` commands.

To turn on Solaris Resource Manager, enabling the SHR scheduler and resource limits:

```
# srmadm set -f /var/srm/srmDB fileopen=y:share=y:limits=y
```

To set the CPU usage decay rate to have a half-life of 5 minutes:

```
# srmadm set usagedecay=300s
```

To display the current flag settings and charges:

```
% srmadm
```

To show all the default settings:

```
% srmadm show -dv
```

Disabling Solaris Resource Manager

The `srmadm`(1MSRM) command can disable Solaris Resource Manager by clearing the `fileopen` flag: all processes are moved onto the surrogate `root` lnode, other changed lnodes in the cache are flushed to disk, and the limits database is closed. This

automatically forces the `share` and `limits` flags off, disabling the SHR scheduler and limit enforcement, respectively. The `share` and `limits` flags may be turned off independently if required while leaving the limits database open. This is better than closing the file, because processes can stay attached to their correct lnodes.

Note that if the Solaris Resource Manager scheduler alone is disabled in mid-operation, all this does is suspend the usage and decay algorithm. The scheduler still continues handling processes in the SHR scheduling class, but as each is assigned an updated priority, the same value is used, resulting in simple “round-robin” scheduling.

Re-enabling Solaris Resource Manager by opening the file and setting the `share` and/or `limits` flags after the file has been closed will not cause existing processes to move off the `root` lnode. It is best not to close the Solaris Resource Manager database during normal operation. If it is closed, the system should be rebooted in order to ensure correct attachment of processes to lnodes.

Using `limdaemon`

By default, `limdaemon` logs messages using `syslog`(3C). A timestamp is included on messages.

`limdaemon` has several options that can be configured when it is started:

- The `-m` tag and `-p` priority options are used to tag the messages and control message routing according to the `syslogd`(1M) configuration.
- The `-c` option causes `limdaemon` to suppress the updating of terminal connect-time usages.
- The `-d` option causes `limdaemon` to decay connect-time usages for all terminals of logged in users, with the interval between decays being the argument of the `-t` option (the default is 1 minute).
- The `-D`*n* option causes `limdaemon` to decay connect-time usages for the terminals of all users once every *n* minutes.
- The `-k` option terminates the currently running `limdaemon` command.
- The `-t` option is used to set the time period (in minutes) between updates to the connect-time usage attribute in the terminal device category. The default is 1 minute.
- The `-e` option can be used to suppress the logging off of users who have reached their connect-time limit. This option is implied by the use of the `-c` option.
- The `-w` option sets the number of minutes before expiration of connect-time that the warning message is given. The default warning interval is 5 minutes.

- The `-g` option is used to set the grace time (in seconds). The default grace time is 30 seconds.

In the following example, the `limdaemon` command:

```
% limdaemon -g300
```

starts the daemon and sets the grace time to 5 minutes. Note that it is not necessary to follow the command with a shell '&' character. When `limdaemon` is started, it makes itself into a daemon. That is, a child process is forked that detaches itself from the controlling terminal, placing itself in a process group of its own.

The administrator should determine the balance needed between the additional overhead incurred for rapid updating of connect-time usage attributes, and the greater granularity that will appear with less frequent updating. See the `limdaemon`(1MSRM) man page for more information on these and other options.

PAM Subsystem

Beginning with the Solaris 2.6 release, Solaris systems support the Pluggable Authentication Module (PAM). Whenever a user requests an operation that involves changing or setting the user's identity (such as logging in to the system, invoking an 'r' command such as `rcp` or `rsh`, using `ftp`, or using `su`), a set of configurable modules are used to provide authentication, account management, credentials management, and session management. Solaris Resource Manager provides a PAM module for login accounting and modifying the behavior of `su`.

The program used to request the operation is called a service.

The PAM system as a whole is documented in the man pages `pam`(3), `pam.conf`(4), `pam_unix`(5), and `pam_srm`(5SRM).

In Solaris Resource Manager, the PAM module provides account management and session management functions. The behavior of PAM can be controlled by editing the file `/etc/pam.conf`. For normal Solaris Resource Manager behavior, the Solaris Resource Manager PAM module should be configured as `requisite` for all login-like services for session management, and as `requisite` for account management for all PAM services. Usually, the Solaris Resource Manager module should be placed after all other `required` and `requisite` modules, and before any other `sufficient` or `optional` modules.

Upon installation, Solaris Resource Manager edits `/etc/pam.conf` to provide a suitable behavior. It inserts lines such as these for each service (including `other`) that already has session or account management configured:

```
login account requisite pam_srm.so.1 nolnode=/etc/srm/nolnode
other session requisite pam_srm.so.1
other account requisite pam_srm.so.1 nolnode=/etc/srm/nolnode
```

The first line specifies that for service `login`, the module `pam_share.so.1` is to be used to provide account management functionality, that it must allow login if `login` is to succeed, and it is to be given the argument `nolnode=/etc/srm/nolnode`. See `pam.conf`(4) for a full explanation of the various control flags (`required`, `requisite`, `optional`, and `sufficient`).

The second line says that the `login` service will use the `pam_share.so.1` module for session management.

The full list of supported arguments for Solaris Resource Manager account and session management modules is found in `pam_srm`(5SRM).

Account Management

When the Solaris Resource Manager account management PAM module gets control:

1. It determines if Solaris Resource Manager is installed and enabled, and tells the PAM system to ignore this module if it is not.
2. It determines whether the user has an lnode, and calls an administrator-configurable 'no lnode' script if not.
3. It determines whether the user has permission to use the requested service and device.
4. It determines whether the user has exceeded the warnings limit, and refuses permission to log in if this is the case.
5. It calls an administrator-configurable 'every login' script.

If any of these steps fail, the remainder are not performed, and the Solaris Resource Manager account management PAM module denies use of the service. An explanatory message is passed to the user through the service where possible.

Scripts

The default 'missing lnode' script will create an lnode for the user and send mail notifying the system administrator that this has been done. The default script is `/etc/srm/nolnode`, but this can be changed by editing the file `/etc/pam.conf` and changing the value of the `nolnode` option on Solaris Resource Manager account

management module lines. The 'every login' script is not usually configured. However, it can be configured by adding an [everylogin=pathname] option to any Solaris Resource Manager account management module in `/etc/pam.conf`. Scripts are invoked as the `root` user. Standard input, output, and error are closed. If a script exits non-zero, access will be denied. All information is passed as environment variables, which are derived directly from information passed to PAM from the service.

`USER` The login name supplied to the program. It has been authenticated by looking it up in the password map; if not present, the account management module will already have returned an error code to PAM.

`UID` The UID of the user being authenticated. For services that change UID (such as `su`), this is the UID of the user invoking the service; for services that set UID (such as `login`), this is the target UID (that of `USER`).

`RHOST` For access attempts across a network, this variable contains the name of the host where the attempt originated. Its value is otherwise implementation dependent.

`SERVICE` The name of the access service, for example, `rsh`, `login`, and `ftp`.

`TTY` The name of the TTY on which the service is being invoked. Some services that do not (strictly speaking) have a controlling terminal (such as `ftp`) will fill this variable with process information (for example, *ftp12345*, where *12345* is the process identifier (PID) of `ftpd`; others leave it empty or replace it with the service name.

`DEBUG` If `debug` was specified in the `pam.conf` file, DEBUG is set to true; otherwise it is set to false. No other environment variables are set, so any script must set its own PATH variable if required.

The default 'no lnode' script creates the lnode in the default scheduling group (`other` if such a user exists in the password map, otherwise `root`) and mails the system administrator a reminder to move the new lnode into the appropriate place in the scheduling hierarchy. For a sample script, see "Default 'no lnode' Script" on page 137.

PAM Interaction With Device Groups

The Solaris Resource Manager PAM module looks up the terminal and service names in the device hierarchy, and returns a 'permission denied' message to its invoker if limits are exceeded or if a device flag evaluates to `set`.

The device categories examined are `terminal` for the terminal name, and `services` for the kind of service requested. For example, an `rlogin` attempt may try to use a file in the network device group, so the flags tested for the user (assuming all flags are set to `group`) are as shown below. These flags are checked in order:

- `terminal.flag.network`
- `terminal.flag.all`

- `services.flag.rlogin`
- `services.flag.netservices`

Access is permitted only if they all evaluate to `set`. In addition, limits will be checked for the corresponding categories (`terminal` and `services`).

Session Management

For login-like services (those that create an entry in the `utmp` file), the session management facilities of PAM are invoked as well as the account management facilities if both are configured in `/etc/pam.conf`.

The Solaris Resource Manager product's session management handles charging for devices. It looks to see if the user has exceeded the connect-time limit, or has the `onelogin` flag evaluate to `set` and is already logged in, and if so, prevents login.

Otherwise, it generates a message to the `limdaemon` process to inform it of the login and the configured cost for the terminal being used. It then informs the kernel that the current process is a login header process, and that the `limdaemon` process must be notified when it expires.

The `limdaemon` process then tracks connect-time limits and issues warnings if they are about to be exceeded.

CHAPTER 4

Boot Procedure

During the Solaris boot procedure, various facets of the Solaris Resource Manager software are enabled at different points. The major steps are presented here:

- When the kernel starts, various parameters are loaded from the `/etc/system` file. Some of these affect Solaris Resource Manager. These are documented in the next section, "Booting Without Solaris Resource Manager" on page 45.
- As the kernel continues its initialization, after process 0 has been created, but before process 1 is started, Solaris Resource Manager is initialized by starting `init` in the Solaris Resource Manager CPU scheduling class (SHR) instead of the default scheduling class. The SHR module is loaded and process 1 (the `init` process) is scheduled by the Solaris Resource Manager software. (See `init`(1M).)
- Initially the `init` process and all its children are attached to the surrogate `root` lnode.
- When the kernel is fully initialized, the system will transition from single-user to one of the multi-user modes (usually run-level 2 or 3). Early in this procedure, the `/etc/init.d/init.srm` script is run. The actions performed by this script are described in "Boot Sequence Events" on page 46 and enable normal Solaris Resource Manager operations.

Booting Without Solaris Resource Manager

If you must boot the system without Solaris Resource Manager active, change the *initclass* variable in the `/etc/system` file to refer to timesharing (TS) instead of SHR. A simple way of doing this is to use the `-a` (ask) option of the `boot` command, so that you will be prompted for a system file. For other prompts, press the Return key to

accept the default values until you are prompted for the name of the system file. At the prompt for the name of the system file, type `etc/system.noshrload` (no leading slash) as the response. Here is an example of the procedure:

```
ok boot -a
Booting from: sd(0,0,0) -a
Enter filename [kernel/unix]:
Enter default directory for modules
 [/platform/SUNW,UltraSPARC/kernel /kernel /usr/kernel]:
SunOS Release 5.6 Version ... [UNIX(R) System V Release 4.0]
Copyright (c) 1983-1997, Sun Microsystems, Inc.
Name of system file [etc/system]: etc/system.noshrload
root filesystem type [ufs]:
Enter physical name of root device
 [/sbus@1,f8000000/esp@0,800000/sd@3,0:a]:
```

Note that `/etc/system.noshrload` is a backup copy of `/etc/system` made at the time Solaris Resource Manager was installed. If there have been subsequent edits to `/etc/system`, then `/etc/system.noshrload` should be maintained in parallel so that it differs only by the occurrence of the Solaris Resource Manager modification:

```
# diff /etc/system /etc/system.noshrload
< # enable srm
< set initclass="SHR"
```

Boot Sequence Events

The sequence in which events occur while switching to multi-user mode is particularly important in Solaris Resource Manager. The following sequence of steps correctly establishes the Solaris Resource Manager system:

1. Configure and enable Solaris Resource Manager using the `srmadm` command.

   At this point, the limits database will be opened and the SHR scheduler will be enabled. See "Enabling Solaris Resource Manager Using `srmadm`" on page 48 for information on this process.
2. Assign the 'lost' (`srmlost`) and 'idle' (`srmidle`) lnodes.
3. Start the Solaris Resource Manager daemon.

   See "Starting the Solaris Resource Manager Daemon" on page 49 for information on this procedure.
4. Start other system daemons on an appropriate lnode.

The default script used in Steps 1 through 3 of the above process is shown in the appendix.

System Daemon Processes

Of particular importance is the attachment of daemons (system maintenance processes which normally run permanently) to an lnode other than the `root` lnode. Processes attached to the `root` lnode are scheduled specially and will always be given all the CPU resources they demand, so it is best not to attach any process that is potentially CPU-intensive to the `root` lnode. Attaching daemons to their own lnode allows the central system administrator to allocate them a suitable CPU share.

During the boot procedure, each new process inherits its lnode attachment from its parent process. Since the `init` process is attached to the `root` lnode, so are all subsequent processes. Until the Solaris Resource Manager initialization script is run and the limits database is opened, processes cannot be attached to other lnodes; even then this only happens when a process does an explicit `setuid` system call (using `login`(1), for example) or explicitly asks Solaris Resource Manager to attach to a nominated lnode, like the `srmuser`(1SRM) command does. Running a program with the `setuid` file mode bit set does not change the lnode attachment.

Consequently, all system programs started automatically during system startup will be attached to the `root` lnode. This is often not desirable, since any process attached to the `root` lnode that becomes CPU intensive will severely disrupt the execution of other processes. Therefore, it is recommended that any daemon processes started as part of the boot procedure be explicitly attached to their own lnode by using the `srmuser` command to invoke them. This will not affect their real or effective UIDs.

A possible example is shown here:

```
/usr/srm/bin/srmuser network in.named
```

This could be used to replace the existing invocation of the `named`(1M) daemon in its startup script. This requires that a user account and lnode for *network* be established beforehand.

Enabling Solaris Resource Manager Using `srmadm`

The `srmadm` command allows the administrator to control the operating state and system-wide configuration of Solaris Resource Manager. This command is typically used during transition to run-level 2 or 3 from within the Solaris Resource Manager `init.d`(4) script `/etc/init.d/init.srm`. It is run to ensure that appropriate values for all parameters are set each time the system is booted, and to ensure that the Solaris Resource Manager system will be enabled prior to users having access to the system. The `srmadm` command is also used to administer the global Solaris Resource Manager parameters. See the `srmadm`(1MSRM) man page for a list of the parameters that can be set using `srmadm`. The `srmadm` commands issued in the Solaris Resource Manager `init.d` script will:

- Open the limits database. Up until this point, any processes that are started are attached automatically to a surrogate `root` lnode. The surrogate `root` lnode is used to ensure that there is always an lnode available to connect processes to, regardless of the operational state of Solaris Resource Manager. For this reason, it is important that the limits database be opened before any non-root processes are started. When the limits database is opened, the values in the usage attributes in the surrogate `root` lnode are added into their counterparts in the real `root` lnode. A limitation of this technique is that any net decrease in usage will not be counted. This ensures that usage alterations prior to the limits database being opened are not discarded.
- Enable limit enforcement.
- Set the parameters which control the behavior of the Solaris Resource Manager SHR scheduler, for example, the usage decay rate.
- Enable the SHR scheduler. Prior to this, processes in the SHR scheduling class are scheduled in a simple round-robin fashion and the CPU entitlements set within the Solaris Resource Manager system have no effect.

Refer to "Global Solaris Resource Manager Parameters via `srmadm`" on page 37 for some common invocations of the `srmadm` command.

Starting the Solaris Resource Manager Daemon

The `limdaemon`(1MSRM) program is the Solaris Resource Manager user-mode daemon. It is normally invoked at transition to run-level 2 or 3 as the last step in the Solaris Resource Manager `init.d` script. It shouldn't be confused with the `srmgr` system process (in the SYS class) that is initiated by the kernel. The following `ps`(1) listing shows both these processes:

```
# ps -efc | egrep 'limdaemon|srmgr'
root     4     0  SYS  60 18:42:14 ?        0:05 srmgr
root    92     1  SHR  19 18:42:32 ?        0:41 limdaemon
```

The `limdaemon` program performs the following functions:

- Receives notification messages and delivers them to the terminals of destination users
- Receives login or log out notification messages, maintaining an exact record of all Solaris Resource Manager login sessions currently in progress
- Periodically updates the connect-time usages for all users who have Solaris Resource Manager login sessions currently in progress (optional)
- Detects users who have reached their connect-time limit, kills their processes, and logs them out (optional) after a grace interval
- Logs all actions using `syslog`(3C) to `syslogd`(1M)

When notified of Solaris Resource Manager login sessions, `limdaemon` monitors the terminal connect-time of all users and checks it against their connect-time limits. When limits are nearly reached, users are sent notification messages. Once the expiration time is reached, a further grace period is allowed before all their processes are terminated and they are logged out.

The `limdaemon` program decays connect-time usages. Usage decay for the terminal device category must be performed, if connect-time limits are used. Refer to "Using `limdaemon`" on page 40 for information on `limdaemon` command-line options.

CHAPTER 5

Managing Lnodes

The Solaris Resource Manager system is built around a fundamental addition to the kernel: a per-user structure called an lnode. An lnode is essentially a fixed-size place in which many kinds of per-user data can be stored and updated. For every unique UID defined in the password map, there should be a corresponding lnode. (This refers to every unique UID returned by successive `getpwent`(3C)calls.) An lnode may exist without a corresponding password map entry, but this is not recommended. Lnodes are stored on disk and automatically moved in and out of memory by the kernel. In-memory copies of lnodes that have been changed since they were read from disk are written back as part of the regular system synchronization operations, as well as on demand when the `sync` command is run, or when necessary to free space in the lnode cache for reading in further lnodes.

Lnodes are maintained as a tree hierarchy, with the *central system administrator* as the head of the tree, and other users as group headers of smaller groups of users within the tree. The central administrator is the superuser, or root user of the system.

Errors relating to lnodes, such as orphans and group loops, are discussed in Chapter 11.

Delegated Administration

The primary responsibility for the administration of lnodes rests with the central administrator. While Solaris Resource Manager introduces several resource controls that may be assigned and managed, it also allows certain administrative privileges to be selectively assigned to non-root users, thereby distributing the task of user administration.

Administrative privileges may be assigned to appropriate users by setting the user's *uselimadm* or *admin* flag. A sub-administrator is a user with a set *uselimadm* flag who has the same `limadm` program administrative privilege as the superuser. A group header with a set *admin* flag is called a group administrator, and has privileges (as described below) over users within the same scheduling group.

The central administrator controls the overall division of the system's resources by creating and assigning limits to scheduling groups who have root as their parent. Group administrators typically perform the same types of resource control, but limited to users within their scheduling group. The division of resources by the group administrator is limited to the resources that have been allocated to the group (for example, those allocated to the group header lnode). Note that group administrators may assign an *admin* flag to any user in their scheduling group, further sub-dividing the administrative responsibilities.

Group administrators can do the following:

1. Alter the resource limits of any user within their scheduling group.

   Note that even though a group administrator can set the limit of a resource to be greater than that of the limit for the group, resources consumed by group members are also considered to be consumed for group headers, and limits on individual users will be enforced when an attempt is made to exceed the group header limit.
2. Alter any flag or attribute (except `flag.uselimadm` and `cpu.usage`) of any lnode within their scheduling group.

   Flag assignments by group administrators are further constrained in that a user cannot be given a privilege that is not already held by the group administrator. This restriction is applied to prevent a group administrator from circumventing the security within Solaris Resource Manager.

A group administrator's main tools are the `limadm`(1MSRM) and `limreport`(1SRM) commands. The `limadm` program performs operations on the limits, flags, and other Solaris Resource Manager attributes of one or more existing users. Combined with the report generator, `limreport`, these tools allow a scheduling group to be autonomously self-managed without disturbing the resource allocations or management of other, disjoint scheduling groups.

The superuser is exempt from all resource limits, always has full administrative privileges regardless of its flag settings, can add, delete, and change user accounts and is able to change any usage, limit, or flag value of any lnode by using the `limadm` command.

Security

Solaris Resource Manager has a wide effect on the administration of a Solaris system, so it is important that it be installed and maintained in a manner that ensures the system is secure.

There are a number of ways in which the system administrator can ensure that the security of the Solaris Resource Manager system is maintained. The most important, as with any Solaris system, is to ensure the privacy of the `root` password. Anyone who knows the `root` user password has unrestricted access to the system's resources, the same as the central administrator.

A number of special administrative privileges can be granted to users within Solaris Resource Manager by setting certain system flags within their respective lnodes. These can help increase the security of a system because they allow delegated users to carry out the tasks that are required of them without giving them full superuser privileges.

Some of these privileges should not be granted lightly because they give the recipient user broad-ranging powers. The passwords of users possessing special privileges should be protected diligently, just as the superuser password should be protected.

There are several security precautions taken within Solaris Resource Manager to prevent misuse of the administrative privilege granted to sub-administrators: Refer to "A Typical Application Server" on page 109 and "Lnode Maintenance Programs" on page 58.

There are circumstances in which the central administrator can leave the system open to security breaches if not careful with the manipulation of the structure of the scheduling tree. It is important for the central administrator to know how to correctly modify the scheduling tree and how to detect potential problems in the current structure. This is discussed in "Scheduling Tree Structure" on page 65.

Suggested Group Administrator Lnode Structure

A problem that group administrators might face is that they share group limits with their group members. For example, if the group header lnode has a process limit set on it, then that limit controls the number of processes that can be used by the entire group, including the group header. Unless further limited, any user within the scheduling group can prevent the group administrator from being able to create new processes simply by exceeding the process limit.

One way to prevent this is for the group administrator to set individual limits on each of the group members. However, to be effective, these limits might have to be overly restrictive. Also, forcing a group administrator to manage individual limits is at odds with the Solaris Resource Manager goal of hierarchical resource control.

An alternate way of solving this problem is for the central administrator to change the structure of the lnodes within the group. Rather than placing users directly beneath the group administrator's lnode, a "control" lnode is created below the group administrator's lnode as the only child lnode, and then users are made children of the control lnode. This results in the structure shown.

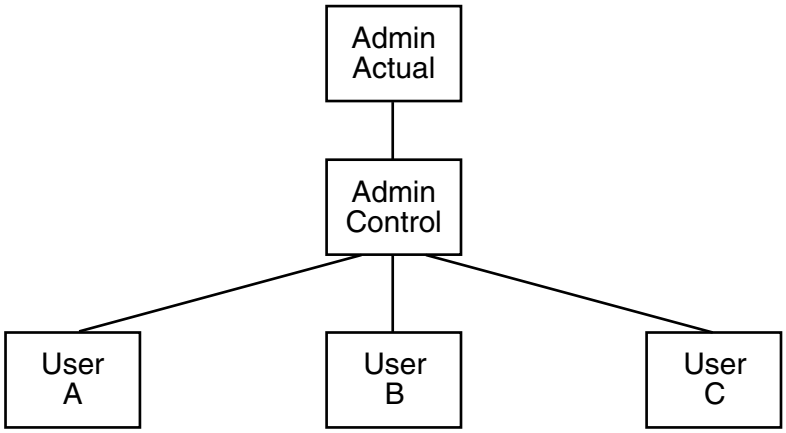


FIGURE 5–1 Group Administrator Lnode Structure

Referring to the previous figure, the UID of the group administrator's account would correspond to that of the lnode labelled "Actual," the parent of the tree. This is the lnode that would have the `admin` flag set. A dummy account would be created for the "Control" lnode. No login need be permitted on this account. The lnodes labelled "A," "B," and "C" correspond to users under the group administrator's control.

In this case, the process limit for the "Actual" lnode could be 100, while that of the "Control" lnode could be 90, with limits for individual users set to 0. This setup would ensure that even if users A, B, and C were using a total of 90 processes (all they are allowed), the sub-administrator can still create 10 processes.

However, it is still possible in this case for users to stop each other from creating processes. The only way to prevent this is to set individual limits on those users. In this example, those limits could be set to 40 each, still allowing flexibility while preventing a single user from completely shutting out the others.

Also note that in this example the central administrator could create extra lnodes for new users as children of the "Control" lnode without having to re-balance limits.

Limits Database

The *limits database* is the database of user information that the Solaris Resource Manager software uses to perform all resource control. It contains one lnode per UID, which is accessed by using the UID as a direct index into the file. If there is an lnode for a numerically large UID, the limits database will appear to be quite large.

However, where the UIDs of users in the system are not sequential, the limits database will have large gaps, or holes, and on a file system type that supports it, may be stored as a sparse file. This means that no disk blocks are actually allocated for storage of the "empty" sections of the file. `ufs` file systems support sparse files, but `tmpfs` file systems do not. See “Saving and Restoring the Limits Database” on page 55 for the implications of sparse files on saving and restoring the limits database.

Whenever you create a new user, you must create a new lnode.

Creating the Limits Database

The Solaris Resource Manager startup file (`/etc/init.d/init.srm`) will create an initial limits database when invoked for the first time or at any boot if the file is missing.

The limits database typically resides in the `/var/srm` directory.

The limits database should be owned by `root`, group owned by `root`, and readable only by the owner. Write permission is not required since only kernel code with superuser credentials writes to the file.

Caution – If a user can write to the Solaris Resource Manager limits database, system security may be compromised.

Saving and Restoring the Limits Database

Because the limits database can be a sparse file, be careful when copying it. The file will most likely consume a lot of disk space if it is written by a utility that does not support sparse files, since the empty regions of the file will read as sequences of zeros and be written back out as real blocks instead of empty regions. This could happen if the file were being copied, backed up, or restored by a utility such as `tar`(1), `cpio`(1), or `cp`(1), although programs such as `ufsdump`(1M) and `ufsrestore`(1M) will preserve holes.

You can also back up and restore the limits database by using `limreport` to generate an ASCII version of the file and using `limadm` to re-create the original file from that saved ASCII version. For example, the command:

```
# limreport 'flag.real' - lname preserve > /var/tmp/savelnodes
```

will create `/var/tmp/savelnodes` as an ASCII representation of the lnodes for each user in the password map. Note that this will not save lnodes for which there is no corresponding password map entry. At most, lnodes should exist for the set of all UIDs in the password map.

The command:

```
# limadm set -f - < /var/tmp/savelnodes
```

recreates the lnodes for which data was saved. This command will not delete lnodes that were not saved, so these techniques can also be used to save and restore selections of lnodes rather than the whole limits database.

"The `limreport` and `limadm` Commands" on page 61 describes the use of the `limreport` and `limadm` commands in more detail. It is helpful for the administrator to be familiar with using these commands to save and restore lnodes, since it may be necessary to use them when a change to the interpretation of the lnode structure (as defined by the limits database) is made.

Because the contents of the limits database are changing regularly during normal system operation, perform backup operations while the system is quiescent, or in single-user mode. Similarly, restore an entire limits database only when the Solaris Resource Manager is not in use, such as when the system is in single-user mode.

Creating and Deleting Lnodes

Whenever a new user is created, a corresponding lnode should be created and its limits and privileges should be set. When using Solaris Resource Manager, the administrator should maintain the limits database in parallel with the normal Solaris password map. The command:

```
# limreport \!flag.real - uid lname
```

can be used to print a list of the UIDs and login names of any users who do not have corresponding lnodes.

Lnodes are not automatically created and deleted by the system commands used to create and delete accounts. It is up to the administrator to perform these actions. However, lnodes can be automatically created on-demand when the user logs in; see "PAM Subsystem" on page 41 for more details.

Similarly, just before a user account is deleted from the password map, the corresponding lnode should be removed from the limits database by using the `limadm`(1MSRM) command.

Note – When deleting lnodes, ensure that sub-trees are deleted from the bottom-most lnodes up. If you start at the top of the sub-tree you are deleting, you will lose control of the children of the lnodes deleted because they will become orphaned when their parents are removed.

If the UID of a user is ever changed, the contents of the user's lnode should be copied to a new lnode corresponding to the new UID and the original lnode should be deleted. See "Copying and Removing Lnodes" on page 62.

Any child lnodes should be attached either to the newly created lnode or to some other suitable parent lnode. The command:

```
# limreport 'sgroup==X' '%u\tsgroup=Y\n' uid | limadm set -u -f -
```

can be used to find all lnodes with a scheduling group parent whose UID is X, and make them children of the lnode with a UID of Y.

The following steps illustrate how to change the UID of an lnode from X to Y.

1. Save the state of the lnode in which the UID is to be changed:

```
# limreport 'uid==X' - lname preserve > /var/tmp/savelnode.X
```

2. Change the UID of the password map entry for the user from the old value (X) to that of the new UID (Y).
3. Create an lnode for the new UID, restoring the state from that which was previously saved:

```
# limadm set -f /var/tmp/savelnode.X
```

4. For all child lnodes of the lnode to be changed (UID X), change their scheduling group to the new lnode (UID Y):

```
# limreport 'sgroup==X' '%u\tsgroup=Y\n' uid | limadm set -u -f -
```

5. Ensure there are no processes currently attached to the old lnode.
6. Use the `chown`(2) command to change the owner of all files owned by the original UID to that of the new UID. For example:

```
# find / -user X -print | xargs chown Y
```

7. Delete the old lnode:

```
# limadm delete X
```

Lnode Maintenance Programs

The `limadm` command is the primary tool available to administrators for maintaining a user's lnode. This command changes Solaris Resource Manager attribute values for a given list of user accounts. If an lnode does not exist for any of the users, then a default-filled blank one is created first. New lnodes are created with the following properties:

- *flag.real* is set;
- *cpu.shares* and *cpu.myshares* attributes are set to 1;
- The flags *uselimadm* and *admin* are set to clear;
- All other flags are set to *inherit*;
- All limit and usage attributes are set to zero.

The scheduling group of the new lnode is set to user 'other' (`srmother`) if an lnode for that user account exists, or else to the `root` lnode.

The `limadm` invoker needs sufficient administrative privilege to perform the specified changes. The invoker must be the superuser, have a set *uselimadm* flag, or be a group administrator who is only changing the attributes of members of the scheduling group to which the invoker belongs. Restrictions apply to the use of `limadm` by group administrators.

- Group administrators cannot change the value of their own attributes.
- A user's `sgroup` attribute can only be assigned to the invoker or to a member of the invoker's scheduling group.
- Group administrators cannot change the attributes of users outside their scheduling group.

- They cannot alter the value of any attribute used to store `usages` other than term usage. Without this restriction, group administrators could circumvent the group limits in their own lnodes by reducing the usage of one of their children, thereby reducing the group usage.
- If they have a flag that evaluates to a value other than the default, group administrators can alter the value of that flag for another member of their group only if they are changing it to be the same non-default value.

  This ensures that group administrators with explicitly denied privileges cannot grant those privileges to any users under their influence.

The `limadm` command allows an administrator to remove an lnode without deleting the corresponding user account in the password map. To use `limadm`, the invoker must be the superuser or have a set *uselimadm* flag. If the invoker only has a set *admin* flag, then the invoker can only modify the lnodes of users under scheduling groups for which the invoker is the group header.

Units

Values within Solaris Resource Manager are represented in one of three types of *units*:

Scaled	The scaled unit is the default. It's an easily readable format used to display and enter values. Scaled units help users avoid making entry errors by reducing the number of digits that need to be entered.
Raw (or unscaled)	The raw unit is the basic unit in which a value is represented. For example, the raw units for virtual memory usage are bytes, and the raw units for virtual memory accrual are byte-seconds. These are mainly employed when billing for usage, when exact quantities are required.
Internal	The internal unit is used by Solaris Resource Manager to store memory attributes in machine-dependent units rather than in bytes.

Conversions

Solaris Resource Manager programs carry out conversions to and from the internal units used to store attribute values, so that the user is always presented with scaled units or raw units. This means that, with few exceptions, the user never need be concerned with the internal units used by Solaris Resource Manager.

The terms exa, peta, tera, giga, mega, and kilo are used within Solaris Resource Manager to represent powers of 2, not powers of 10. For example, a megabyte is 1,048,576 bytes, not 1,000,000 bytes. The powers of 2 for each term are 60 (exa), 50 (peta), 40 (tera), 30 (giga), 20 (mega), and 10 (kilo).

The programs that are the primary interface between users and the Solaris Resource Manager system are `limadm`, `liminfo`, and `limreport`. The conversions and scaling that they carry out are detailed in the following subsections.

The `limadm` Command

When changing attribute values, `limadm` allows numbers to be suffixed by scale characters: *[EPTGMK][B][.][wdhms]*. Uppercase and lowercase are interchangeable.

If the attribute has the dimension of storage (memory attributes) or of storage accrual, then a character from the first group (EPTGMK) is allowed. This multiplies by the number of bytes in 1 exabyte (E), petabyte (P), terabyte (T), gigabyte (G), megabyte (M), or kilobyte (K) as appropriate. The optional B character may be appended for user readability, but it has no effect.

If the attribute has the dimension of time (type date or time), or of storage accrual, then a character from the second group is allowed. This multiplies by the number of seconds in one week (w), day (d), hour (h), minute (m), or second (s) as appropriate.

An optional period may separate the storage and time units (for example. mh, M.h and MB.h all stand for 'megabyte hours').

Where ambiguity exists in the use of the M suffix, `limadm` attempts to derive its meaning from the context. If this is not possible, it is assumed to mean mega, not minutes.

When inputting large numbers, these conversion characters are useful to avoid errors in the order of magnitude of the entry, but the quantity is stored in internal units regardless of the method of entry.

A special scale character `u` can also be used, by itself, but only for memory attribute values. It indicates that the number is in machine-dependent (internal) units instead of bytes.

The `liminfo` Command

The `liminfo`(1SRM) command uses the same suffixes when reporting as `limadm` uses for input (see above). Normally, `liminfo` converts values into appropriate scaled formats to be printed, but the `-r` option can be used to cause `liminfo` to print values

in their raw (unscaled) form. For example, memory is normally scaled to a suitable unit, such as megabytes (for example, '102 MB'), but specifying the `-r` option causes it to be printed in bytes (for example, 106954752 bytes).

The `limreport` Command

The `limreport`(1SRM) command always reports values in their raw (unscaled) form. If scaled values are required, the conversion must be stated explicitly in the expression used to display the value. For example, to display total virtual memory usage for all users in kilobytes, rounded up to the nearest kilobyte:

```
# limreport 'flag.real' '%-8.8s %d KB\n' lname '(memory.usage+1k-1)/1k'
```

As this example demonstrates, you can use the scaling suffixes on numbers in expressions, which simplifies the conversion of raw units to scaled values.

Note that the internal units for some attributes are not the same as their 'raw' form. Normally, this does not concern the user because all the Solaris Resource Manager programs carry out conversion to scaled units or raw units. However, it does mean that, for example, select-expressions in `limreport` that specify an exact match on a number of bytes will always fail to match if a number is specified that is not an integral multiple of the relevant internal unit.

Manipulating Lnodes

The `limreport` and `limadm` Commands

The `limreport` and `limadm` commands provide the administrator with an easy way to save and restore the contents of lnodes for any number of users. Use the `limreport` command to select and extract the lnodes that are to be saved, and use `limadm` to restore them. This combination of commands is most commonly used for copying lnodes and for altering the lnode structure, as discussed in the following sections.

The `limreport` command also provides a flexible way to select and display users' attributes. It provides two levels of selection: selection of lnodes, and selection of attributes to be displayed for each lnode selected. The lnode selection is achieved by specification of a select-expression, which may be a single condition or a set of

conditions joined by logical operators in a C-style syntax. The attribute selection is achieved by listing the attributes' symbolic names. The way in which the attributes will be displayed is specified by a format control string, similar to the C function `limreport`, with extensions to handle special Solaris Resource Manager types. If a format control string of '-' is specified, `limreport` uses default formats for each attribute displayed. Refer to `limreport`(1SRM) for further details.

Changing the Lnode Structure

The `limadm` command provides a facility to indivisibly change the contents of attributes within lnodes, given that the invoker has sufficient privilege. Change commands can be specified directly on the command line, or the name of a file containing the change commands can be specified (by using the `-f` option).

The `limreport` command is able to generate attribute value assignments using the `limadm` syntax (refer to the *preserve* identifier in the `limadm` syntax), the output of which can be input to `limreport` using the `-f` option. This allows the administrator to use the two programs together to selectively save and restore the contents of the limits database.

Copying and Removing Lnodes

The command:

```
# limreport 'uid==X' - Y preserve | limadm set -u -f -
```

will copy an lnode from UID X to UID Y. The expression `uid==X` provides the method for selecting the source lnode. The *preserve* identifier causes `limreport` to output all attribute values that are not read-only in a syntax that is suitable to pass to `limadm`. Placing the UID Y prior to the *preserve* identifier causes this to be the first item in the data passed to the `limadm`, thus providing the selection of the target lnode.

If the source lnode is no longer required, it can be removed using `limadm`.

Note – Be careful when using a match by UID as the `limreport` selection expression. If multiple login names share a UID, they will all be matched. In the example above, this would not matter; the same lnode data will be preserved and loaded multiple times. In the Solaris environment, UID 0 has login names of both `root` and `smtp`.

CHAPTER 6

SHR Scheduler

The Solaris Resource Manager SHR scheduler is used to control the allocation of the CPU resource. The concept of shares allows administrators to easily control relative entitlements to CPU resources for users, groups, and applications. The concept of shares is analogous to that of shares in a company; what matters is not how many you have, but how many compared with other shareholders.

Technical Description

There are four attributes per lnode associated with the Solaris Resource Manager CPU scheduler: `cpu.shares`, `cpu.myshares`, `cpu.usage`, and `cpu.accrue`. The output of `liminfo`(1SRM) displays these attributes and other useful values.

Solaris Resource Manager scheduling is implemented using the SHR scheduling class. This includes support for the `nice`(1), `priocntl`(1), `renice`(1), and `dispadmin`(1M) commands. At the system-call level, SHR is compatible with the TS scheduling class.

Shares

A user's `cpu.shares` attribute is used to apportion CPU entitlement with respect to the user's *parent* and active peers. A user's `cpu.myshares` attribute is meaningful only if the user has *child* users who are active; it is used to determine the proportion of CPU entitlement with respect to them.

For example, if users A and B are the only children of parent P, and A, B, and P each have one share each within group P (that is, A and B have `cpu.shares` set to 1, while P has `cpu.myshares` set to 1), then they each have a CPU entitlement of one-third of the total entitlement of the group.

Thus, the actual CPU entitlement of a user depends on the parent's relative entitlement. This, in turn, depends on the relative values of `cpu.shares` of the parent to the parent's *peers* and to the `cpu.myshares` of the grandparent, and so on up the scheduling tree.

For system management reasons, processes attached to the `root` lnode are not subject to the shares attributes. Any process attached to the `root` lnode is always given almost all the CPU resources it requests.

It is important that no CPU-intensive processes be attached to the `root` lnode, since that would severely impact the execution of other processes. To avoid this, the following precautions should be taken:

- The central administrator account should have its own UID, one that is different than the superuser account. This account should be used when logging in to perform non-administrative activities. If a UID of superuser is needed to carry out administrative functions, the central administrator can use the `su`(1) command to change UIDs while still keeping the account attached to its own lnode.
- The `srmuser`(1SRM) command can be used in the `init.d`(4) scripts to attach any daemon processes to a non-`root` lnode. Any processes started in the boot script have, by default, an effective UID of `root`, and they are attached to the `root` lnode. The user command allows daemons to retain an effective UID of `root`, while attached to their own lnode. This will avoid problems if any of the daemons become CPU-intensive.

Not all group headers in the scheduling tree need to represent actual users who run processes, and in these cases it is not necessary to allocate them a share of CPU. Such lnodes can be indicated by setting their `cpu.myshares` attribute to zero. The `cpu.accrue` attribute in such a group header still includes all charges levied on all members of its group.

Allocated Share

The `cpu.shares` and `cpu.myshares` attributes determine each active lnode's current *allocated share* of CPU, as a percentage. The shares of inactive users make no difference to allocated share. If only one user is active, that user will have 100 percent of the available CPU resource. If there are only two active users with equal shares in the same group, each will have allocated shares of 50 percent. See "Calculation of Allocated Share" on page 67 for more information on how the allocated share is calculated.

Usage and Decay

The `cpu.usage` attribute increases whenever a process attached to the lnode is charged for a CPU tick. The usage attribute value exponentially decays at a rate determined by the usage decay global Solaris Resource Manager parameter. The usage decay rate (described by a half-life in seconds) is set by the `srmadm`(1MSRM) command.

Although all processes have an lnode regardless of their current scheduling class, those outside the SHR scheduling class are never charged.

Accrued Usage

The *accrued* usage attribute increases by the same amount as the usage attribute, but is not decayed. It therefore represents the total accumulated usage for all processes that have been attached to the lnode and its members since the attribute was last reset.

Effective Share

An lnode's allocated share, together with its `cpu.usage` attribute, determines its current *effective share*. The Solaris Resource Manager scheduler adjusts the priorities of all processes attached to an lnode so that their rate of work is proportional to the lnode's effective share, and inversely proportional to the number of runnable processes attached to it.

Per-Process Share Priority (`sharepri`)

Each process attached to an lnode has internal data, specific to Solaris Resource Manager, that is maintained by the operating system kernel. The most important of these values for scheduling purposes is the `sharepri` value. At any time, the processes with the lowest `sharepri` values will be the most eligible to be scheduled for running on a CPU.

Sample Share Allocation

Scheduling Tree Structure

The following points relate to the structure of the *scheduling tree*, which is an area requiring special consideration by the central administrator:

- The scheduling tree is the structure used by Solaris Resource Manager to implement a hierarchy of resource and privilege control. If a sub-administrator gains control over a sub-tree of the scheduling tree that the sub-administrator would normally not have access to, that person can gain access to additional resource usage and privileges without the approval of the central administrator. One way for this to happen is if an administrator removes an lnode and leaves an orphaned sub-tree behind.
- The central administrator can use the `limreport`(1SRM) command to identify orphaned sections of the scheduling tree by employing the built-in orphan identifier. Any orphans found should then immediately be reattached.
- When a new lnode is created, it is mostly zero-filled, which causes most flags to have the default value of *inherit*. This is the desired effect for most flags, because they are used to indicate device privileges. There are two flags that are explicitly cleared at lnode creation time—the `uselimadm` and `admin` flags. This is to prevent new users from automatically gaining any administrative privilege.

Description of Tree

The tree shown below defines a structure consisting of several group headers and several ordinary users. The top of the tree is the `root` user. A group header lnode is shown with two integers, which represent the values of its `cpu.shares` and `cpu.myshares` attributes, respectively. A *leaf lnode* is shown with a single integer, which represents the value of its `cpu.shares` attribute only.

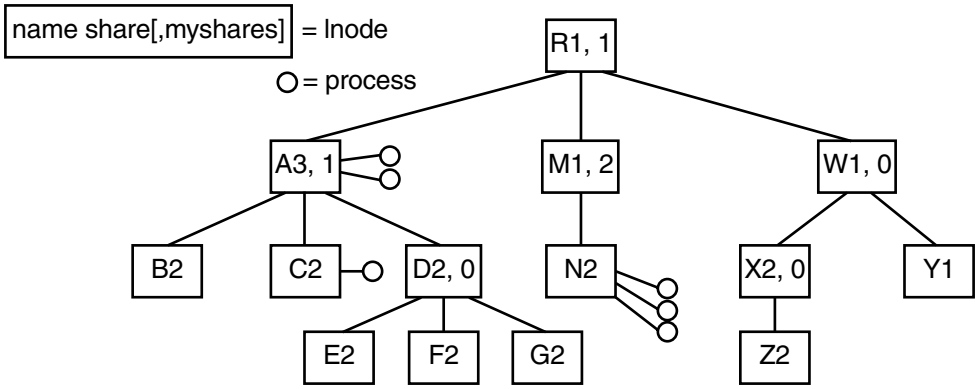


FIGURE 6–1 Scheduling Tree Structure

Calculation of Allocated Share

Using the previous figure as an example, nodes A, C, and N currently have processes attached to them. At the topmost level, the CPU would only need to be shared between A and M since there are no processes for W or any member of scheduling group W. The ratio of shares between A and M is 3:1, so the allocated share at the topmost level would be 75 percent to group A, and 25 percent to group M.

The 75 percent allocated to group A would then be shared between its active users (A and C), in the ratio of their shares within group A (that is, 1:2). Note that the *myshares* attribute is used when determining A's shares with respect to its children. User A would therefore get one third of the group's allocated share, and C would get the remaining two thirds. The whole of the allocation for group M would go to lnode N since it is the only lnode with processes.

The overall distribution of allocated share of available CPU would therefore be 0.25 for A, 0.5 for C, and 0.25 for N.

Further suppose that the A, C, and N processes are all continually demanding CPU and that the system has at most two CPUs. In this case, Solaris Resource Manager will schedule them so that the individual processes receive these percentages of total available CPU:

- For the two A processes: 12.5 percent each
- For the C process: 50 percent
- For the three N processes: 8.3 percent each

The rate of progress of the individual processes is controlled so that the target for each lnode is met. On a system with more than two CPUs and only these six runnable processes, the C process will be unable to consume the 50 percent entitlement, and the residue is shared in proportion between A and N.

Solaris Resource Manager and the Solaris `nice` Facility

The `nice` facility in the Solaris environment allows a user to reduce the priority of a process so that normal processes will not be slowed by non-urgent ones. With Solaris Resource Manager, the incentive for users to use this facility is a reduced charge rate for CPU time used at a lower priority.

Solaris Resource Manager implements this effect by allowing the central administrator to bias the `sharepri` decay rate for processes which have applied `nice`. The `pridecay` global Solaris Resource Manager parameter in the `srmadm`(1MSRM) command is used to set the decay rates for the priorities of processes with normal and maximum `nice` values. The rates for all intervening `nice` values are interpolated between them and similarly extrapolated to the minimum `nice` value. For example, the priority (for example, `sharepri`) for normal processes may be decayed with a half-life of 2 seconds, while the priority of processes with a maximum `nice` value may be decayed with a half-life of 60 seconds.

The effect is that processes using `nice` to reduce their priority get a smaller share of CPU than other processes on the same lnode. Under Solaris Resource Manager `nice` has little influence on execution rates for processes on different lnodes unless the queue of runnable processes exceeds the number of CPUs.

Solaris Resource Manager treats processes with a maximum `nice` value (for example, those started with a `nice -19` command) specially. Such processes will only be granted CPU ticks if no other process requests them and they would otherwise be idle.

For information on `nice`, see `nice`(1) and `nice`(2SRM). For information on the relationship of Solaris Resource Manager to other resource control features, see "Differences Between Solaris Resource Manager and Similar Products" on page 22.

Dynamic Reconfiguration

The dynamic reconfiguration (DR) feature of Sun Enterprise servers enables users to dynamically add and delete system boards, which contain hardware resources such as processors, memory, and I/O devices. Solaris Resource Manager keeps track of the available processor resources for scheduling purposes and appropriately handles the changes, fairly redistibuting currently available processor resources among eligible users and processes.

Because Solaris Resource Manager controls only the virtual memory sizes of processes, not the physical memory used by processes and users, the effect of a DR operation on memory has no impact on Solaris Resource Manager's memory-limit checking.

srmidle Lnode

The *idle lnode* (srmidle) is the lnode assigned by the central administrator to charge for all the kernel's idle CPU costs. At installation, srmidle was created with a UID of *41*. The srmidle lnode should have zero shares, to ensure that the processes attached to it are run only when no other processes are active. The srmidle lnode is assigned using the srmadm command.

At boot time, the default idle lnode is the root lnode. At transition to multi-user mode, the init.d script will set the idle lnode to that of the account srmidle if such an account exists. This behavior can be customized by specifying a different lnode to use in the /etc/init.d/init.srm script.

If the idle lnode is not root, then it must be a direct child of root.

srmother Lnode

The *other lnode* (srmother) is the lnode assigned by the system administrator as the default parent lnode for new users created after the initial install (where root is the default parent lnode). The srmother lnode, which is created automatically by the system at installation time and cannot be changed, has a default value of 1 share, to ensure that lnodes attached to it will have access to the CPU. The srmother lnode was created with a UID of *43*.

The srmother lnode should have no resource limits, a CPU share of 1 or more, and no special privileges.

srmlost Lnode

Under Solaris Resource Manager, the setuid(2SRM) system call has the side effect of attaching the calling process to a new lnode. If the change of attachment fails, typically because the new lnode does not exist, the process is attached instead to the *lost lnode* (srmlost), which was created when you installed Solaris Resource Manager. If this attachment also fails or no srmlost lnode has been nominated, then the setuid function is unaffected and the process continues on its current lnode.

The `init.srm` script sets the `srmlost` lnode during the transition to multi-user mode. This behavior can be overridden by specifying an lnode to use in the `/etc/init.d/init.srm` file. To avoid security breaches, the `srmlost` lnode should have a CPU share of 1, and no special privileges. If you alter the values, consider the requirements for this user when making the change.

The `srmlost` lnode was created with a UID of *42*.

CHAPTER 7

Memory Limits, Process Memory Limits, and Process Count Limits

Solaris Resource Manager allows an administrator to control:

- The amount of virtual memory held by users or an application (this is a measure at the lnode level of all processes' virtual memory usage)
- The amount of virtual memory held by a group of users or an application (this is a measure at the group header lnode level of all processes' virtual memory usage where those processes are attached to the group header and/or to its children)
- The amount of virtual memory held by an individual process
- The number of processes that a user/lnode can run concurrently

Attributes for Control of Virtual Memory (Total)

The system attributes used to record virtual memory usage and assign limits to it are:

`memory.myusage` The `memory.myusage` attribute of an lnode is equal to the sum of the virtual memory usage of all processes currently attached to the lnode.

`memory.usage` The `memory.usage` attribute of an lnode is equal to the sum of its `memory.myusage` attribute and the `memory.usage` attributes of its child lnodes.

`memory.limit` The `memory.limit` attribute is applied to the `memory.usage` attribute; it limits the combined memory usage of all processes attached to the lnode and all member lnodes.

Attribute for Control of Process Memory (Per-Process)

The system attribute used to record process usage and assign limits to it is:

`memory.plimit` The `memory.plimit` attribute of an lnode is a per-process limit that is applied separately to the memory usage of each process attached to it or to any of its members.

Technical Description of Memory Limits

Whenever a process attempts to increase its memory size, it is subject to the memory limits (total and per-process) of the lnode to which it is attached. There are five means by which a process can attempt to increase its memory size:

1. Calling an allocation routine, such as `malloc`(3C), which results in an invocation of the `sbrk`(2) system call. If a memory limit is exceeded, the call will return an error with errno set to ENOMEM.
2. Expanding the stack and causing a stack fault, which would normally cause an extra page of memory to be given to the process. If a memory limit is exceeded, the process will be sent a SIGSEGV signal.
3. Using `mmap`(2).
4. Using `fork`(2). The child address space is duplicated while it is still owned by the parent process. During duplication, the new address space will not exceed the `memory.plimit` since the parent must already be within this limit; however, the allocation is subject to the `memory.limit`.
5. Using `exec`(2). During an `exec`, memory usage first decreases as the old address space is discarded. However, if the address space of the new program is larger, and causes a limit to be exceeded, the `exec` can fail.

Dynamic Reconfiguration and Virtual Memory Limits

The dynamic reconfiguration (DR) feature available on the Enterprise 10000, Sun Fire 15K, and Sun Fire 12K systems has limited impact on the virtual memory limits imposed on lnodes. Specifically, DR makes it possible to add or remove physical memory from the system while the system is up and running. (The system's pool of virtual memory includes all of the physical memory, plus the swap space configured into the system.) Additionally, the `swap`(1M) command can be used on any Solaris system to add (`-a`) or delete (`-d`) swap space from the system. Thus, the total amount of virtual memory can grow or shrink during operation.

This has an indirect impact on the virtual memory limits imposed by Solaris Resource Manager. Because virtual memory is managed in absolute units (rather than shares), the effective limits do not change when the system's resources change during operation. (Note that this behavior is different from the dynamic reconfiguration of processors, as discussed in "Dynamic Reconfiguration" on page 68.)

Attributes for Control of Process Count

The system attributes used to administer process usage (the number of processes) are:

`process.myusage` The `process.myusage` attribute of an lnode is equal to the number of processes attached to the lnode.

`process.usage` The `process.usage` attribute of an lnode is equal to the sum of its `process.myusage` attribute and the `process.usage` attributes of its child lnodes.

`process.limit` The `process.limit` attribute is a limit that applies to the `process.usage` attribute of an lnode; it limits the number of processes that can be attached to the lnode and all member lnodes concurrently.

Technical Description of Process Count

The `fork`(2) and `vfork`(2) system calls are used to create new processes. If this would cause the process limit to be exceeded, the system call fails, returning an EAGAIN error. Most programs interpret EAGAIN as meaning a temporary shortage of system resources and try the fork again, perhaps after a short sleep. If the fork failure is due to a Solaris Resource Manager limit, this can lead to looping for an indefinite amount of time because EAGAIN will be returned on each retry until the limit is fixed for the affected lnode.

CHAPTER 8

Physical Memory Management Using the Resource Capping Daemon

Note – This feature is only available if you are using Solaris Resource Manager 1.3 with Solaris 8 *SPARC Platform Edition*.

A resource *cap* is an upper bound placed on the consumption of a resource, such as physical memory or CPU time, by a collection of processes.

In Solaris Resource Manager 1.3, resource cap enforcement enables you to regulate the resource consumption of physical memory by collections of processes attached to an lnode or to a project. For information about projects, see `project`(4).

Attribute for Limiting Physical Memory Usage

A physical memory resource cap can be defined for an lnode or a project (depending on the mode in use) by adding an attribute to the lnode or project.

The lnode attribute used to assign a resident set size (RSS) cap is:

`rss.limit` The `rss.limit` attribute defines the physical memory cap, in specified units, that is applied to collections of processes attached to an lnode. The cap must be a positive number.

`rss.limit` is set through `limadm`(1MSRM). For example, to set a 10–gigabyte RSS cap for the lnode *oracle*, type:

```
# /usr/srm/sbin/limadm set rss.limit=10G oracle
```

The project attribute used to establish an RSS cap is:

`rcap.max-rss` The `rcap.max-rss` attribute defines the total amount of physical memory, in bytes, that is available to processes attached to a project. The cap must be a positive number.

`rcap.max-rss` is configured in the `project`(4) database. For example, the following line in the project database sets an RSS cap for a project named *oracle*.

```
oracle:100::oracle,root::rcap.max-rss=10737418240
```

Technical Description

The resource cap enforcement daemon `rcapd`(1MSRM) and its associated utilities provide mechanisms for resource cap enforcement and administration. Only one instance of `rcapd` can run at any given time. The daemon periodically samples the resource utilization of collections of processes that have physical memory caps defined. If process consumption exceeds the cap quantity and other conditions are met, the daemon takes actions to reduce the processes' aggregated resource consumption to a level below the cap quantity.

The virtual memory system divides physical memory into segments known as pages. To read data from a file into memory, the virtual memory system reads in one page at a time, or *pages in* a file. To reduce resource consumption, the daemon can page out, or relocate, infrequently used pages to an area outside of physical memory.

Use `rcapadm`(1MSRM) to configure the resource capping daemon. `rcapadm` is also used to administer resource caps at a global level. Configuration changes can be incorporated into `rcapd` on demand by sending it `SIGHUP` (see `kill`(1)), or through the configuration interval (see `rcapadm`(1MSRM)). You must have superuser privileges to configure the daemon. If used without arguments, `rcapadm` displays the current status of the resource capping daemon if it has been configured.

Use `rcapstat`(1SRM) to report resource cap enforcement daemon statistics for each cap defined.

Using the Resource Capping Daemon

Use `rcapadm`(1MSRM) to configure the daemon.

Selecting the Mode of Operation

Two modes of operation are supported, `lnode` and `project`.

To enforce caps defined for lnodes and ignore caps defined for projects, type:

```
# rcapadm -m lnode
```

To enforce caps defined for projects and ignore caps defined for lnodes, type:

```
# rcapadm -m project
```

Tuning Operation Intervals

You can tune the intervals for the periodic operations performed by `rcapd`. To reset intervals, use the `-i` option.

```
# rcapadm -i interval=value,...,interval=value
```

All intervals are specified in seconds. The intervals and their default values are described in the following table.

Interval	Default Value in Seconds	Description
`scan`	15	Rate at which `rcapd` scans for new processes. Minimum value is 1 second.
`sample`	5	Rate of process resident set size sampling. Minimum value is 1 second.
`report`	5	Rate at which paging statistics are updated by `rcapd` for `rcapstat`. If set to `0`, statistics will not be updated.
`config`	60	Rate of reconfiguration. In a reconfiguration event, `rcapd` checks its configuration file for updates, and scans lnode (limits) databases or project databases for new lnode or project caps, or new collections.

If the interval specified to `rcapstat` is shorter than the interval specified to `rcapd` (with `rcapadm`(1MSRM)), the output for some intervals can be zero. This is because `rcapd` does not update statistics more frequently than the interval specified with `rcapadm`, and this interval is independent of, and less precise than, the sampling interval used by `rcapstat`.

Setting Memory Cap Enforcement Value

Caps can be configured so that they will not be enforced until the physical memory available to processes is low. The minimum (and default) value is `0`, which means that memory caps are always enforced. To set a different minimum physical memory utilization for memory cap enforcement, type:

```
# rcapadm -c percent
```

The *percent* value should be in the range 0 to 100. The current global physical memory utilization and its cap can be obtained by using the `-g` option to the `rcapstat` command. See "Reporting Global Memory Caps" on page 83.

Enabling Resource Capping

To enable the resource capping daemon so that it will be started now and also be started each time the system is booted, type :

```
# rcapadm -E
```

To enable the resource capping daemon at boot without affecting its current running status, also specify the `-n` option:

```
# rcapadm -n -E
```

Disabling Resource Capping

To disable the the resource capping daemon so that it will be stopped now and not be started when the system is booted, type :

```
# rcapadm -D
```

To disable the resource capping daemon without affecting its current running status, also specify the `-n` option:

```
# rcapadm -n -D
```

Note – If killed, `rcapd` can leave processes in a stopped state. See `kill`(1). Use `rcapadm -D` or `SIGTERM` to cause `rcapd` to terminate properly.

Monitoring the Resource Capping Daemon

Use the `rcapstat` command to issue reports on lnodes or projects capped by `rcapd`. You can set the sampling interval and specify the report frequency.

interval Specifies the sampling interval in seconds. The default interval is 5 seconds.

count Specifies the number of times that the statistics are repeated. By default, `rcapstat` reports statistics until a termination signal is received or until the `rcapd` process exits.

The paging statistics in the first report issued by `rcapstat` show the activity since the daemon was started. Subsequent reports reflect the activity since the last report was issued.

The following list defines the column headings in an `rcapstat` report.

`id` The lnode ID of the collection of processes or project ID of the processes attached to a project.

`lnode/project` The collection ID type, which is `project` or `lnode`.

`nproc` The number of processes in the collection.

`vm` The total virtual memory size of the collection of processes, including all mapped files and devices, in kilobytes (K), megabytes (M), or gigabytes (G).

`rss` The total resident set size (RSS) of the collection of processes, in kilobytes (K), megabytes (M), or gigabytes (G), not accounting for objects that are shared.

`cap` The RSS cap defined for a project or an lnode. See `rcapd`(1MSRM) for information about how to specify memory caps.

`at` The total amount of memory that `rcapd` attempted to page out since the last `rcapstat` sample.

`avgat` The average amount of memory that `rcapd` attempted to page out during each sample cycle that occurred

	since the last `rcapstat` sample. The rate at which `rcapd` samples collection RSS can be set with `rcapadm`(1MSRM).
`pg`	The total amount of memory that `rcapd` successfully paged out since the last `rcapstat` sample. `rcapd` pages out the most infrequently used pages.
`avgpg`	An estimate of the average amount of memory that `rcapd` successfully paged out during each sample cycle that occurred since the last `rcapstat` sample. The rate at which `rcapd` samples process RSS sizes can be set with `rcapadm`.

Producing Reports With `rcapstat`

The examples in this section show you how to use `rcapstat` to monitor the resource utilization of collections of processes that have physical memory caps defined.

Using `rcapstat` to Report Cap and Lnode Information

Caps are defined for two lnodes associated with two users. `user1` has a cap of 50 megabytes, and `user2` has a cap of 10 megabytes.

The following command produces reports at 5-second sampling intervals. A report will be issued five times, once after each sample.

```
usermachine% rcapstat 5 5
    id lnode   nproc    vm   rss   cap    at avgat    pg avgpg
112270 user1      24  123M   35M   50M   50M    0K 3312K    0K
 78194 user2       1 2368K 1856K   10M    0K    0K    0K    0K
    id lnode   nproc    vm   rss   cap    at avgat    pg avgpg
112270 user1      24  123M   35M   50M    0K    0K    0K    0K
 78194 user2       1 2368K 1856K   10M    0K    0K    0K    0K
    id lnode   nproc    vm   rss   cap    at avgat    pg avgpg
112270 user1      24  123M   35M   50M    0K    0K    0K    0K
 78194 user2       1 2368K 1928K   10M    0K    0K    0K    0K
    id lnode   nproc    vm   rss   cap    at avgat    pg avgpg
112270 user1      24  123M   35M   50M    0K    0K    0K    0K
 78194 user2       1 2368K 1928K   10M    0K    0K    0K    0K
    id lnode   nproc    vm   rss   cap    at avgat    pg avgpg
112270 user1      24  123M   35M   50M    0K    0K    0K    0K
 78194 user2       1 2368K 1928K   10M    0K    0K    0K    0K
```

The first three lines of output constitute the first report, which contains the cap and lnode information for the two lnodes, and paging statistics since `rcapd` was started. The `at` and `pg` columns are a number greater than zero for `user1` and zero for `user2`, which indicates that at some time in the daemon's history, `user1` exceeded its cap but `user2` did not.

The subsequent reports contain paging statistics since the prior intervals, but show no significant activity.

Lowering the Cap of an Lnode

The `limadm`(1MSRM) command can be used to lower the memory cap of an lnode, which makes the cap more restrictive. `rcapd` will enforce the new cap after the next configuration interval (see `rcapadm`(1MSRM)). A signal can also be sent, which causes `rcapd` to enforce the new cap immediately.

```
admin# limadm set rss.limit=30M user1
admin# pkill -HUP rcapd
```

The following command produces reports at 5-second sampling intervals. A report will be issued five times, once after each sample.

```
admin# rcapstat 5 5
    id lnode   nproc    vm   rss   cap    at avgat    pg avgpg
112270 user1      24  123M   35M   30M   50M    0K 3312K    0K
 78194 user2       1 2368K 1856K   10M    0K    0K    0K    0K
    id lnode   nproc    vm   rss   cap    at avgat    pg avgpg
112270 user1      24  123M   36M   30M   52M   52M  632K  632K
 78194 user2       1 2368K 2096K   10M    0K    0K    0K    0K
    id lnode   nproc    vm   rss   cap    at avgat    pg avgpg
112270 user1      24  123M   33M   30M   57M   52M  816K  632K
 78194 user2       1 2368K 1968K   10M    0K    0K    0K    0K
    id lnode   nproc    vm   rss   cap    at avgat    pg avgpg
112270 user1      24  123M   27M   30M 4792K 4792K   40K   40K
 78194 user2       1 2368K 1144K   10M    0K    0K    0K    0K
    id lnode   nproc    vm   rss   cap    at avgat    pg avgpg
112270 user1      24  123M   27M   30M    0K    0K    0K    0K
 78194 user2       1 2368K 1144K   10M    0K    0K    0K    0K
```

When the cap was lowered to 30 megabytes from 50 megabytes, `rcapd` responded by attempting to page out the 6-megabyte amount of resident memory above the cap value. The goal was reached; it was exceeded by a small amount.

Using `rcapstat` to Monitor the RSS of a Project

The following command produces reports at 5-second sampling intervals. A report will be issued five times, once after each sample.

```
user1machine% rcapstat 5 5
```

```
id     project   nproc    vm   rss   cap    at avgat    pg avgpg
376565 user1        57  209M   46M   10M  440M  220M 5528K 2764K
376565 user1        57  209M   44M   10M  394M  131M 4912K 1637K
376565 user1        56  207M   43M   10M  440M  147M 6048K 2016K
376565 user1        56  207M   42M   10M  522M  174M 4368K 1456K
376565 user1        56  207M   44M   10M  482M  161M 3376K 1125K
```

In this example, the project `user1` has an RSS in excess of its physical memory cap. The nonzero values in the `pg` column indicate that `rcapd` is consistently paging out memory as it attempts to meet the cap by lowering the physical memory utilization of the project's processes. However, `rcapd` is unsuccessful, as indicated by the varying `rss` values that do not show a corresponding decrease. This means that the application's resident memory is being actively used, forcing `rcapd` to affect the working set. Under this condition, the system will continue to experience high page fault rates, and associated I/O, until the working set size is reduced, the cap is raised, or the application changes its memory access pattern.

Determining the Working Set Size of a Project

The following example is a continuation of the previous example, and it uses the same project.

```
example% rcapstat 5 5
    id project    nproc    vm   rss   cap    at avgat    pg avgpg
376565 user1         56  207M   44M   10M  381M  191M   15M 7924K
376565 user1         56  207M   46M   10M  479M  160M 2696K  898K
376565 user1         56  207M   46M   10M  424M  141M 7280K 2426K
376565 user1         56  207M   43M   10M  401M  201M 4808K 2404K
376565 user1         56  207M   43M   10M  456M  152M 4800K 1600K
376565 user1         56  207M   44M   10M  486M  162M 4064K 1354K
376565 user1         56  207M   52M  100M  191M   95M 1944K  972K
376565 user1         56  207M   55M  100M    0K    0K    0K    0K
376565 user1         56  207M   56M  100M    0K    0K    0K    0K
376565 user1         56  207M   56M  100M    0K    0K    0K    0K
376565 user1         56  207M   56M  100M    0K    0K    0K    0K
376565 user1         56  207M   56M  100M    0K    0K    0K    0K
```

By inhibiting cap enforcement, either by raising the cap of a project or by changing the minimum cap enforcement memory pressure value (see `rcapadm`(1MSRM)), the resident set can become the working set. The `rss` column might stabilize to show the project working set size, as shown in this example. This is the minimum cap value that will allow the project's processes to operate without perpetually incurring page faults.

Reporting Global Memory Caps

You can use the `-g` option of `rcapstat` to report the following:

- Global memory utilization cap set by `rcapadm`(1MSRM)
- Current memory utilization as a percentage of all physical memory installed on the system

References

For additional information, see `rcapadm`(1MSRM), `rcapd`(1MSRM), `rcapstat`(1SRM), `limadm`(1MSRM), and `project`(4).

CHAPTER 9

Usage Data

Solaris Resource Manager provides the administrator with a precise mechanism for collecting accrued usage values for CPU, application, and user resources. This information, plus the functions and utilities provided with Solaris Resource Manager, can be used as a base for the development of a resource billing system.

Accrue Attributes

An lnode's accrue attributes are used to store information about the accrual of resource usage. For example, an lnode's `cpu.accrue` attribute contains the accrued CPU usage for all lnodes within the group as well as that of the current lnode. When any of an lnode's accrue attributes are updated, the change is also applied to the lnode's parent (as with changes to the usage attribute) and so on up to the root lnode, so that the accrued usage at each level in the scheduling tree is the sum of the accrued usage for the lnode and the accrued usage of its children, if any.

For example, a system with two workloads, A and B, might have an lnode hierarchy with lnodes A and B reporting to the root node, and individual users reporting into nodes A and B. The CPU usage of each user can be observed through the usage fields in the lnode of each user, but the sum total of CPU used for workload A is available by looking at the accrued CPU usage in the lnode for workload A.

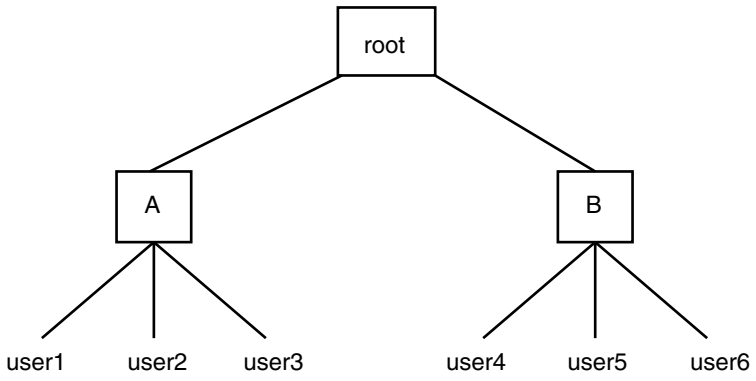


FIGURE 9–1 Accounting Based on the Workload Hierarchy

Billing Issues

The administrator must decide which lnodes are to be billed for resource usage. For example, administrators may only be concerned with billing entire departments. Thus, they may only want to bill the group headers of the *topmost groups*, whose accrued usage will include all the accrued usage of the lnodes at lower levels within their departments.

For administrators to be able to implement a billing system, they must determine a costing function for each resource to be billed. This can be a simple linear relationship (where unit cost is the same, regardless of the amount used), or it can be a non-linear relationship, such as a step function, or a curve where unit cost varies as the amount of usage varies.

In deciding upon a costing function for each resource, the administrators should keep in mind that the costing function will not only control the assignment of cost to accrued resource usage, it can also have an impact on the way in which a user uses the resource. For example, if the costing function for virtual memory usage causes the unit cost to increase as the amount of usage increases, there is a strong incentive for users to keep virtual memory usage low. Therefore, it is possible for administrators to control user behavior through the use of an appropriate costing strategy.

There is only one accrue attribute per resource. It contains the accrued usage for the resource based on the usage attribute for the resource. This means that there is no accrued usage corresponding to the `myusage` attribute. For group headers, there is no accrued usage for the user as an individual, because the accrue attribute holds the group's accrued usage. For lnodes with no children (leaf lnodes), this does not matter,

since the myusage attribute and the usage attribute are identical. If a bill is required for the individual accrued usage for a group header, it must be calculated from the group total less the sum of the individual totals of each child in the group.

The liminfo Command

The default output of the liminfo command is designed for users who want to find out their current usages, limits, and privileges. liminfo is also of use to administrators who want to inquire on the attributes of other users. There are a number of different report formats that can be requested, including the use of options that make the output of liminfo suitable for processing by a pipeline of filters. Refer to liminfo(1SRM) for details of the command-line options and their meanings, and for a description of the fields that can be displayed.

The limreport Command

The limreport command allows an administrator to query any attributes, including the accrue attributes, of any user. The command provides a flexible way to select information to be displayed from the chosen lnodes.

For example, the command:

```
% limreport 'cpu.accrue!=0' '%u %s %f\n' uid lname cpu.accrue
```

selects all lnodes with any accrued usage in the usr domain, and lists the UID and accrued usage attribute from each of the selected lnodes. To sort these values by cpu.accrue and list only the top 10 users, pipe the result to a sort command:

```
% limreport 'cpu.accrue!=0' '%u %s %f\n' uid lname cpu.accrue | sort -2n | head
```

Refer to limreport(1SRM) for additional information.

The `limadm` Command

The `limadm` command can be used within a billing system to zero the accrue attributes after they have been billed. For example, the command:

```
# limreport 1 '%u\tcpu.accrue=0,mem.accrue=0\n' uid | limadm set -u -f -
```

uses the `limreport` command to generate a list of commands piped to `limadm`. Every lnode is selected, and for each lnode, the accrue attribute is zeroed.

The administrator should take care in determining when to clear the accrue attribute of an lnode. The timing will depend on the billing strategy. For example, if bills are to be produced at a group level, and then individual bills are to be produced for the group members, the accrue attributes of the group members should not be cleared until after both bills have been produced. However, if individual bills are not to be produced, the group members' accrue attributes should be cleared at the same time as the group header's, even though they may not have been individually used.

Refer to `limadm`(1MSRM) for additional information.

CHAPTER 10

Advanced Usage

This chapter further explores ways to prioritize and manage dissimilar applications on the system. An example that illustrates these capabilities and other key concepts associated with the Solaris Resource Manager software is provided. The last part of this chapter discusses how to configure Solaris Resource Manager in a Sun Cluster 3.0 12/01 (and later update) environment.

Batch Workloads

Most commercial installations have a requirement for batch processing. Batch processing is typically done at night, after the daily online workload has diminished. This is usually the practice for two reasons: to consolidate the day's transactions into reports, and to prevent batch workloads from impacting the online load.

See "Examples" on page 98 for an illustration of a hierarchy to control the environment in which batch jobs are run; this section also covers the Solaris Resource Manager commands used in the process.

Resources Used by Batch Workloads

Batch workloads are a hybrid between online transaction processing (OLTP) and decision support system (DSS) workloads, and their effect on the system lies somewhere between the two. A batch workload can consist of many repetitive transactions to a database, with some heavy computational work for each. A simple example would be the calculation of total sales for the day. In this case, the batch process would retrieve every sales transaction for the day from the database, extract the sales amount, and maintain a running sum.

Batch processing typically places high demands on both processor and I/O resources, since a large amount of CPU is required for the batch process and the database, and a large number of I/Os are generated from the backend database for each transaction retrieved.

A batch workload is controlled by effectively limiting the rate of consumption of both CPU and I/O. Solaris Resource Manager allows fine-grained resource control of CPU, but I/O resources must be managed by allocating different I/O devices to each workload.

Two methods are typically used to isolate batch resource impact:

- Make a copy of the database on a separate system and run the batch and reporting workloads on that separate system. (Note, however, that in most situations, the batch process updates parts of the online database and cannot be separated from it.)
- Use CPU resource control.

Because the amount of I/O generated from a batch workload is proportional to the amount of CPU consumed, limits on CPU cycles can be used to indirectly control the I/O rate of the batch workload. Note, however, that care must be taken to ensure that excessive I/O is not generated on workloads that have very light CPU requirements.

Problems Associated With Batch Processing

By definition, a batch workload is a workload that runs unconstrained, and it will attempt to complete in the shortest time possible. This means that batch is the worst resource consumer, because it will take all the resources it needs until it is constrained by a system bottleneck (generally the smallest dataflow point in the system).

Batch presents two problems for system managers; it can impact other batch jobs running concurrently, and it can never be run together with the online portion of the workload during business hours.

Even if the batch jobs are scheduled to run during off-hours, for example, from 12:00 a.m. to 6:00 a.m., a system problem or a day of high sales could cause the batch workload to spill over into business hours. Although not quite as bad as downtime, having a batch workload still running at 10:30 a.m. the next day could make online customers wait several minutes for each transaction, ultimately leading to fewer transactions.

Using resource allocation will limit the amount of resources available to the batch workloads and constrain them in a controlled manner.

Consolidation

Solaris Resource Manager permits system resources to be allocated at a system wide level, sometimes in proportion with business arrangements of machine usage between departments. “Examples” on page 98 shows how the Solaris Resource Manager hierarchy is used to achieve this.

Virtual Memory and Databases

The Solaris Resource Manager product also provides the capability to limit the amount of virtual memory used by users and workloads. This capability does not manage physical memory; rather it effectively restricts the amount of global swap space that is consumed by each user.

When a user or workload reaches the virtual memory limit set for their lnode, the system returns a memory allocation error to the application; that is, calls to `malloc()` fail. This error code is reported to the application as though the application had run out of swap space.

Few applications respond well to memory allocation errors. Thus, it is risky to ever let a database server reach its virtual memory limit. In the event that the limit is reached, the database engine might crash, resulting in a corrupted database.

Virtual memory limits should be set high so they are not reached under normal circumstances. Additionally, the virtual memory limit can be used to place a ceiling over the entire database server, which will stop a failing database with a memory leak from affecting other databases or workloads on the system.

Managing NFS

NFS, Sun’s distributed computing environment, runs as kernel threads and uses the kernel scheduling class SYS. Since scheduling allocation for NFS is not managed by the Solaris Resource Manager SHR class, no CPU resource control of NFS is possible. The Solaris Resource Manager product’s ability to allocate processor resources may be reduced on systems offering extensive NFS service.

NFS can, however, be controlled by using network port resource management. For example, Solaris Bandwidth Manager can be used to control the number of NFS packets on the server. NFS can also be managed in some cases by using processor sets to limit the number of CPUs available in the system class.

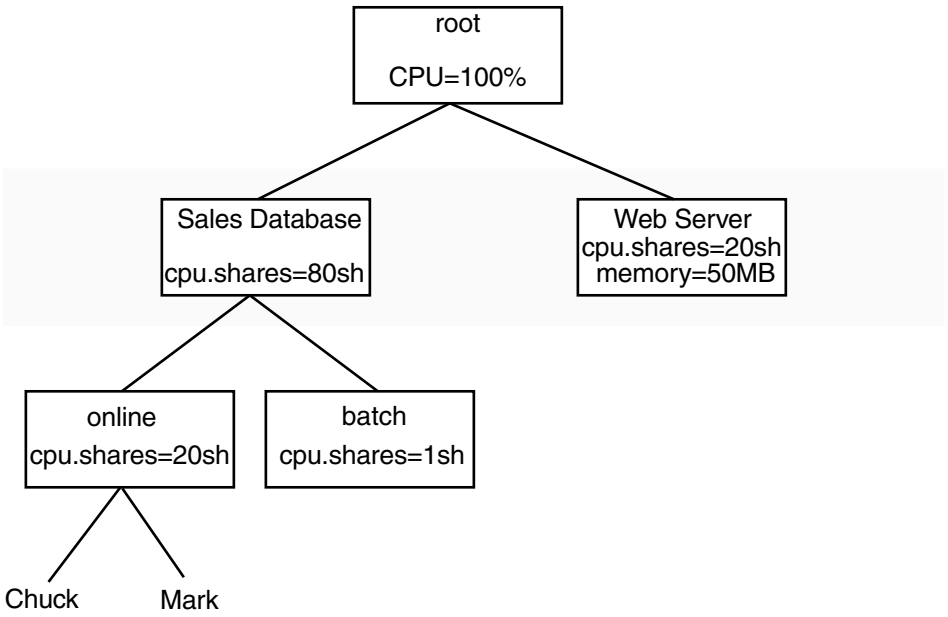
Managing Web Servers

The Solaris Resouce Manager software can be used to manage resources on web servers by controlling the amount of CPU and virtual memory. Three basic topologies are used on systems that host web servers.

Resource Management of a Consolidated Web Server

A single web server can be managed by controlling the amount of resource that the entire web server can use. This is useful in an environment in which a web server is being consolidated with other workloads. This is the most basic form of resource management, and simply prevents other workloads from impacting the performance of the web server, and vice versa. For example, if a CGI script in the web server runs out of control with a memory leak, the entire system will not run out of swap space; only the web server will be affected.

In this example, a web server is allocated 20 shares, which means that it is guaranteed at least 20 percent of the processor resources should the database place excessive demands on the processor.

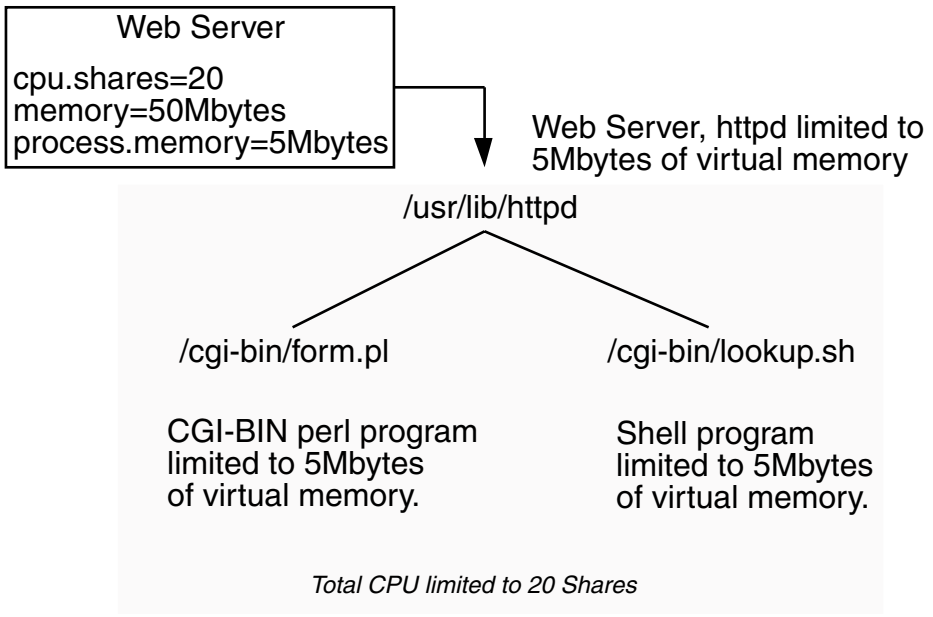


See “Putting on a Web Front-end Process ” on page 102 for an additional web server example.

Finer-Grained Resource Management of a Single Web Server

There are often requirements to use resource management to control behavior within a single web server. For example, a single web server can be shared between many users, each with their own `cgi-bin` programs.

An error in a single `cgi-bin` program can cause the entire web server to run slow, or in the case of a memory leak, could even bring down the web server. To prevent this from happening, per-process limits can be used.



Resource Management of Multiple Virtual Web Servers

Single machines are often used to host multiple virtual web servers in a consolidated fashion. In this case, multiple instances of the `httpd` web server process exist, and there is far greater opportunity to exploit resource control through Solaris Resource Manager.

It is possible to run each web server as a different UNIX UID by setting a parameter in the web server configuration file. This effectively attatches each web server to a different lnode in the Solaris Resource Manager hierarchy.

For example, the Sun WebServer™ has the following parameters in the configuration file `/etc/http/httpd.conf`:

```
# Server parameters
 server  {
   server_root                  "/var/http/"
   server_user                  "webserver1"
   mime_file                    "/etc/http/mime.types"
   mime_default_type            text/nlain
   acl_enable                   "yes"
   acl_file                     "/etc/http/access.acl"
   acl_delegate_depth           3
   cache_enable                 "yes"
```

```
  cache_small_file_cache_size   8                       # megabytes
  cache_large_file_cache_size   256                     # megabytes
  cache_max_file_size           1                       # megabytes
  cache_verification_time       10                      # seconds
  comment                       "Sun WebServer Default Configuration"

  # The following are the server wide aliases

  map   /cgi-bin/               /var/http/cgi-bin/              cgi
  map   /sws-icons/             /var/http/demo/sws-icons/
  map   /admin/                 /usr/http/admin/

# To enable viewing of server stats via command line,
# uncomment the following line
  map   /sws-stats              dummy                           stats
}
```

By configuring each web server to run as a different UNIX UID, you can set different limits on each web server. This is particularly useful for both control and accounting for resource usage on a machine hosting many web servers.

In this case, you can make use of many or all of the Solaris Resource Manager resource controls and limits:

Shares [`cpu.shares`]	The `cpu.shares` can be used to proportionally allocate resources to the different web servers.
Mem limit [`memory.limit`]	The `memory.limit` can be used to limit the amount of virtual memory that the web server can use, which will prevent any one web server from causing another to fail due to memory allocation.
Proc mem limit [`memory.plimit`]	The per-process memory limit can be used to limit the amount of virtual memory a single `cgi-bin` process can use, which will stop any `cgi-bin` process from bringing down its respective web server.
Process limit [`process.limit`]	The maximum total number of processes allowed to attach to a web server can effectively limit the number of concurrent `cgi-bin` processes.

The Role and Effect of Processor Sets

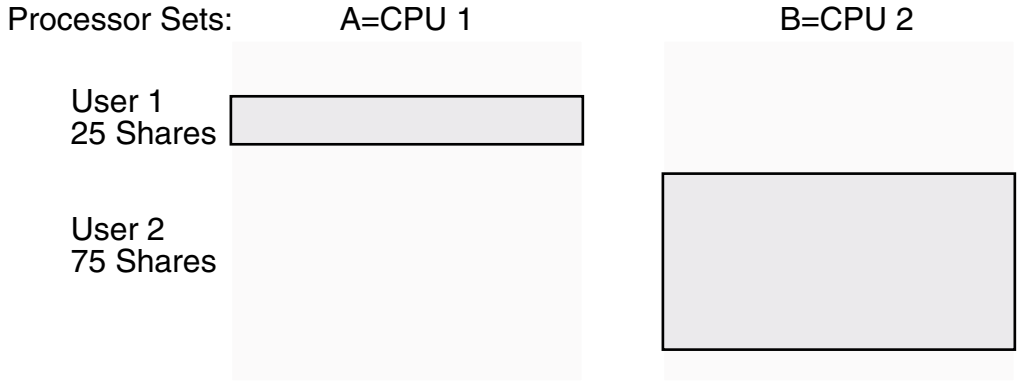
Even with the Solaris Resource Manager software in effect, processor sets can still play an important role in resource allocation. There might be cases in which a system must have hard limits applied to the resource policies. For example, a company may purchase a single 24–processor system, and then host two different business units from the same machine. Each of the business units pays for a proportion of the machine, 40 percent and 60 percent, for example. In this scenario, the administrator might want to establish that the business that pays for 40 percent of the machine never gets more than that share.

With processor sets, it is possible to divide the workloads into 40 percent and 60 percent by allocating 10 processors to the unit with 40 percent, and 14 processors to the unit with 60 percent.

When using processor sets with the Solaris Resource Manager product, it is important to understand the interaction between these two technologies. In some circumstances, the net effect might be different than anticipated.

A Simple Example

The following illustration shows a simple combination of Solaris Resource Manager and processor sets. In this example, processor sets and Solaris Resource Manager CPU shares are mixed.

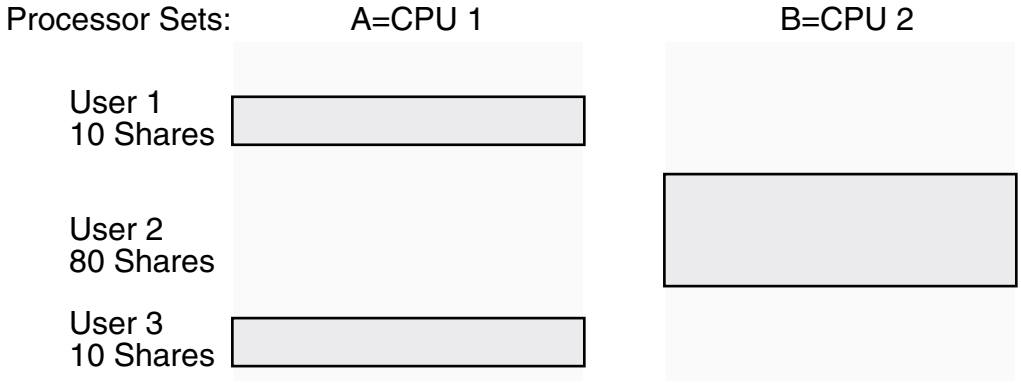


User 1 has 25 Solaris Resource Manager shares and is restricted to processor set A (1 CPU). User 2 has 75 Solaris Resource Manager shares and is restricted to processor set B (1 CPU).

In this example, user 2 will consume its entire processor set (50 percent of the system). Because user 2 is only using 50 percent (rather than its allocated 75 percent), user 1 is able to use the remaining 50 percent. In summary, each user will be granted 50 percent of the system.

A More Complex Example

The following example shows a more complex scenario in which processor sets and Solaris Resource Manager CPU shares are mixed.

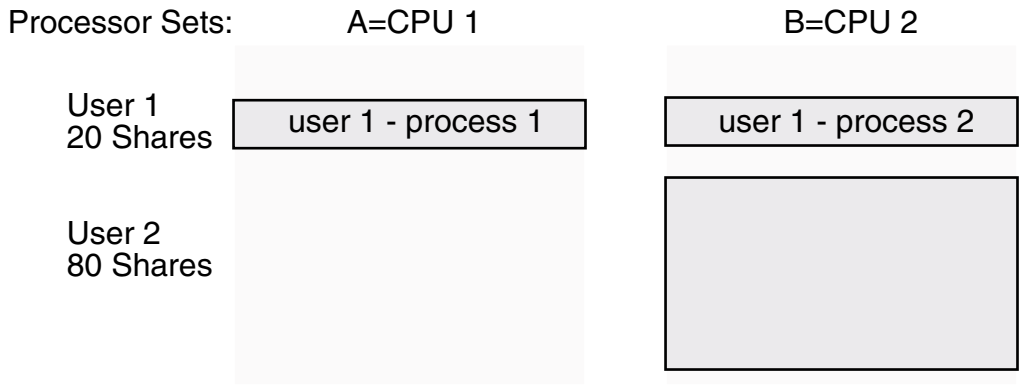


Users 1 and 3 have 10 Solaris Resource Manager shares each and are restricted to processor set A (1 CPU). User 2 has 80 Solaris Resource Manager shares and is restricted to processor set B (1 CPU).

In this example, user 2 will consume its entire processor set (50 percent of the system). Because user 2 is only using 50 percent (rather than its allocated 80 percent), users 1 and 3 are able to use the remaining 50 percent. This will mean that users 1 and 3 get 25 percent of the system, even though they are allocated only 10 shares each.

A Scenario to Avoid

The following scenario should be avoided.



In this scenario, one user has processes in both processor sets. User 1 has 20 Solaris Resource Manager shares and has processes in each processor set. User 2 has 80 Solaris Resource Manager shares and is restricted to processor set B (1 CPU).

In this example, user 1's first process will consume its entire processor set (50 percent of the system). Since user 2 is allowed 80 shares, user 2's process will consume its entire processor set (50 percent). Thus, user 1's second process will get no share of the CPU.

Examples

The examples in this section demonstrate Solaris Resource Manager functions used to control system resources and allocation, and to display information.

Server Consolidation

The first example illustrates these commands:

`liminfo`	Prints user attributes and limits information for one or more users to a terminal window
`limadm`	Changes limit attributes or deletes limits database entries for a list of users
`srmadm`	Displays or sets operation modes and system-wide Solaris Resource Manager tunable parameters
`srmstat`	Displays lnode activity information

Consider the case of consolidating two servers, each running a database application, onto a single machine. Simply running both applications on the single machine results in a working system. Without Solaris Resource Manager, the Solaris system allocates resources to the applications on an equal-use basis, and does not protect one application from competing demands by the other application. However, Solaris Resource Manager provides mechanisms that keep the applications from suffering resource starvation. With Solaris Resource Manager, this is accomplished by starting each database attached to lnodes referring to the databases, *db1* and *db2*. To do this, three new administrative placeholder users must be created, for example, *databases*, *db1*, and *db2*. These are added to the limits database; since lnodes correspond to UNIX UIDs, these must also be added to the `passwd` file (or password map, if the system is using a name service such as NIS or NIS+). Assuming that the UIDs are added to the `passwd` file or password map, the placeholder users *db1* and *db2* are assigned to the *databases* lnode group with the commands:

```
# limadm set sgroup=0 databases
# limadm set sgroup=databases db1 db2
```

which assumes that `/usr/srm/bin` is in the user's path.

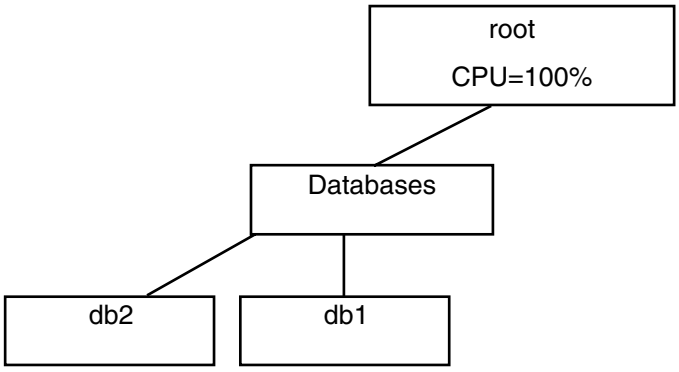


FIGURE 10–1 Server Consolidation

Because there are no other defined groups, the *databases* group currently has full use of the machine. Two lnodes associated with the databases are running, and the processes that run the database applications are attached to the appropriate lnodes with the `srmuser` command in the startup script for the database instances.

```
# srmuser db1 /usr/bin/database1/init.db1
# srmuser db2 /usr/bin/database2/init.db2
```

When either database, *db1* or *db2*, is started up, use the `srmuser` command to ensure that the database is attached to the correct lnode and charged correctly (`srmuser` does not affect the ownership of the process to do this). To run the above command, a user must have the UNIX permissions required to run `init.db1` and the administrative permission to attach processes to the lnode *db1*. As users log in and use the databases, activities performed by the databases are accrued to the lnodes *db1* and *db2*.

By using the default allocation of one share to each lnode, the usage in the *databases* group will average out over time to ensure that the databases, *db1* and *db2*, receive equal allocation of the machine. Specifically, there is one share outstanding—to the *databases* group—and *databases* owns it. Each of the lnodes *db1* and *db2* are also granted the default allocation of one share. Within the *databases* group, there are two shares outstanding, so *db1* and *db2* get equal allocation out of *databases*' resources (in this simple example, there are no competing allocations, so *databases* has access to the entire system).

If it turns out that activity on Database1 requires 60 percent of the machine's CPU capacity and Database2 requires 20 percent of the capacity, the administrator can specify that the system provide at least this much (assuming that the application demands it) by increasing the number of `cpu.shares` allocated to *db1*:

```
# limadm set cpu.shares=3 db1
```

There are now four shares outstanding in the *databases* group; *db1* has three, and *db2* has one. This change is effected immediately upon execution of the above command. There will be a period of settling when the lnode *db1* (Database1) will actually receive more than its entitled 60 percent of the machine resource, as Solaris Resource Manager works to average the usage over the course of time. However, depending on the decay global parameter, this period will not last long.

To monitor this activity at any point, use the commands `liminfo` (see "A Typical Application Server" on page 109) and `srmstat`, in separate windows. Note that `srmstat` provides a regularly updating display. For additional information on `srmstat`, see `srmstat`(1SRM).

You now have a machine running with two database applications, one receiving 75 percent of the resource and the other receiving 25 percent. Remember that `root` is the top-level group header user. Processes running as `root` thus have access to the entire system, if they so request. Accordingly, additional lnodes should be created for running backups, daemons, and other scripts so that the `root` processes cannot possibly take over the whole machine, as they might if run in the traditional manner.

Adding a Computational Batch Application User

This example introduces the following command:

`srmkill` Kills all the active processes attached to an lnode

The Finance department owns the database system, but Joe, a user from Engineering, has to run a computational batch job and would like to use Finance's machine during off hours when the system is generally idle. The Finance department dictates that Joe's job is less important than the databases, and agrees to run his work only if it will not interfere with the system's primary job. To enforce this policy, add a new group (*batch*) to the lnode database, and add Joe to the new *batch* group of the server's lnode hierarchy:

```
# limadm set cpu.shares=20 databases
# limadm set cpu.shares=1 batch
# limadm set cpu.shares=1 joe
# limadm set sgroup=batch joe
```

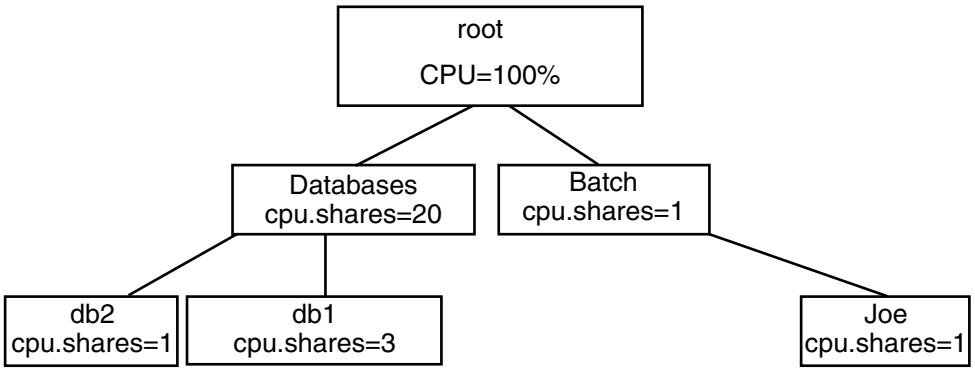


FIGURE 10–2 Adding a Computation Batch Application

This command sequence changes the allocation of shares so that the *databases* group has 20 shares, while the *batch* group has just one. This specifies that members of the *batch* group (only Joe) will use at most 1/21 of the machine if the *databases* group is active. The *databases* group receives 20/21, or 95.2 percent, more than the 60% + 20% = 80% previously determined to be sufficient to handle the database work. If the *databases* are not requesting their full allocation, Joe will receive more than his 4.8 percent allocation. If the *databases* are completely inactive, Joe's allocation might reach 100 percent. When the number of outstanding shares allocated to *databases* is increased from 1 to 20, there is no need to make any changes to the allocation of shares for *db1* and *db2*. Within the *databases* group, there are still four shares outstanding, allocated in the 3:1 ratio. Different levels of the scheduling tree are totally independent; what matters is the ratio of shares between peer groups.

Despite these assurances, the Finance department further wants to ensure that Joe is not even able to log in during prime daytime hours. This can be accomplished by putting some login controls on the *batch* group. Since the controls are sensitive to time of day, run a script that only permits the *batch* group to log in at specific times. For example, this could be implemented with `crontab` entries, such as:

```
0 6 * * * /usr/srm/bin/limadm set flag.nologin=set batch
0 18 * * * /usr/srm/bin/limadm set flag.nologin=clear batch
```

At 6:00 a.m., *batch* does not have permission to log in, but at 18:00 (6 p.m.), the limitation is removed.

An even stricter policy can be implemented by adding another line to the `crontab` entry:

```
01 6 * * * /usr/srm/bin/srmkill joe
```

This uses the `srmkill`(1MSRM) command to kill any processes attached to the lnode Joe at 6:01 a.m. This will not be necessary if the only resources that the job requires are those controlled by Solaris Resource Manager. This action could be useful if Joe's job could reasonably tie up other resources that would interfere with normal work. An example would be a job that holds a key database lock or dominates an I/O channel.

Joe can now log in and run his job only at night. Because Joe (and the entire *batch* group) has significantly fewer shares than the other applications, his application will run with less than 5 percent of the machine. Similarly, `nice`(1) can be used to reduce the priority of processes attached to this job, so it runs at lower priority than other jobs running with equal Solaris Resource Manager shares.

At this point, the Finance department has ensured that its database applications have sufficient access to this system and will not interfere with each other's work. The department has also accommodated Joe's overnight batch processing loads, while ensuring that his work also will not interfere with the department's mission-critical processing.

Putting on a Web Front-end Process

Assume a decision has been made to put a web front-end on Database1, but limit this application to no more than 10 users at a time. Use the process limits function to do this.

First, create a new lnode called *ws1*. By starting the Webserver application under the *ws1* lnode, you can control the number of processes that are available to it, and hence the number of active `http` sessions.

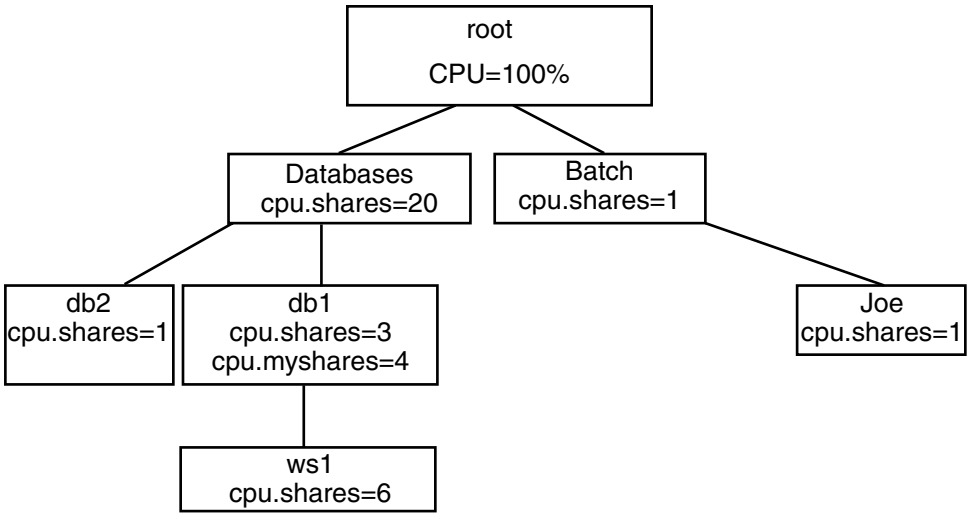


FIGURE 10–3 Adding a Web Front-end Process

Since Webserver is part of the Database1 application, you might want to give it a share of the *db1* lnode and allow it to compete with Database1 for resources. Allocate 60 percent of compute resources to the Webserver and 40 percent to the Database1 application itself:

```
# limadm set cpu.shares=6 ws1
# limadm set sgroup=db1 ws1
# limadm set cpu.myshares=4 db1
# srmuser ws1 /etc/bin/Webserver1/init.webserver
```

The last line starts up the Webserver and charges the application to the *ws1* lnode. Note that for Database1, the `cpu.myshares` have been allocated at 4. This sets the ratio of shares for which *db1* will compete with its child process, Webserver, at a ratio of 4:6.

Note – `cpu.shares` shows the ratio for resource allocation at the peer level in a hierarchy, while `cpu.myshares` shows the ratio for resource allocation at the parent:children level when the parent is actively running applications. Solaris Resource Manager allocates resources based on the ratio of outstanding shares of all active lnodes at their respective levels, where "respective level" includes the `my.shares` of the group parent and all children.

To control the number of processes that Webserver can run, put a process limit on the *ws1* lnode. The example uses 20 since a Webserver query will typically spawn 2 processes, so this in fact limits the number of active Webserver queries to 10:

```
# limadm set process.limit=20 ws1
```

Another application has now been added to the scheduling tree, as a leaf node under an active lnode. To distribute the CPU resource between the active parent and child, use `cpu.myshares` to allocate some portion of the available resource to the parent and some to the child. Process limits are used to limit the number of active sessions on an lnode.

Adding More Users Who Have Special Memory Requirements

This example implements the resource control mechanisms CPU sharing, process limits, and login controls, and it addresses display tools for printing lnodes and showing active lnodes.

`srmadm`	Administers Solaris Resource Manager
`limreport`	Outputs information on selected users
`limdaemon`	Directs daemon to send messages when any limits are reached

Another user, Sally, has also asked to use the machine at night, for her application. Since her application is CPU-intensive, to ensure that Joe's application does not suffer, put a limit on Sally's usage of virtual memory, in terms of both her total usage and her "per-process" usage:

```
# limadm set memory.limit=50M sally
# limadm set memory.plimit=25M sally
```

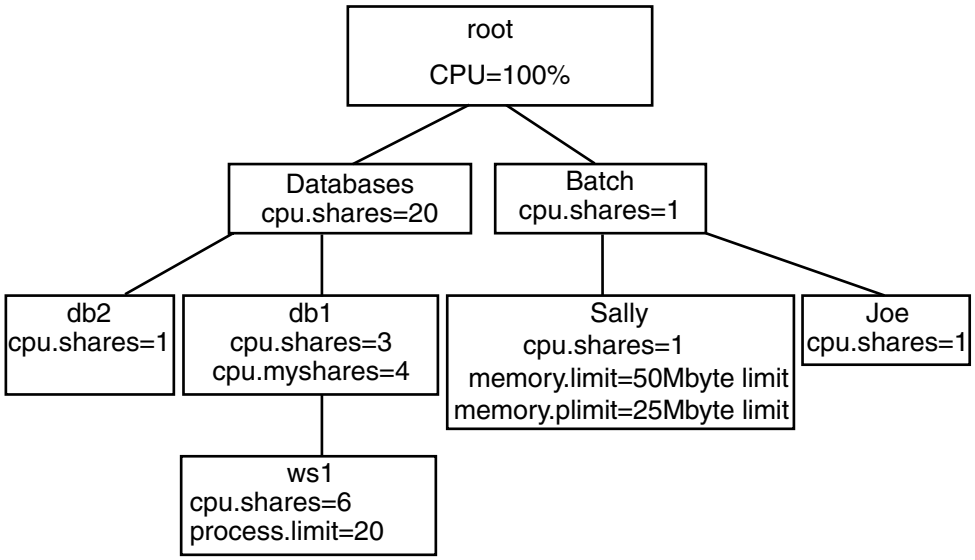


FIGURE 10–4 Adding More Users

If and when Sally's application tries to exceed either her total virtual memory limit or process memory limit, the `limdaemon` command will notify Sally and the system administrator, through the console, that the limit has been exceeded.

Use the `limreport` command to generate a report of who is on the system and their usages to date. A typical use of `limreport` is to see who is using the machine at any time and how they fit within the hierarchy of users:

```
% limreport 'flag.real' - uid sgroup lname cpu.shares cpu.usage |sort +1n +0n
```

Note – `limreport` has several parameters. In this example, a check is made on "`flag.real`" (only looking for "real" lnodes/UIDs); the dash (`-`) is used to indicate that the default best guess for the output format should be used, and the list "`uid sgroup lname cpu.shares cpu.usage`" indicates `limreport` should output these five parameters for each lnode with `flag.real` set to TRUE. Output is piped to a UNIX primary sort on the second column and secondary sort on the first column to provide a simple report of who is using the server.

Anyone with the correct path and permissions can check on the status of Solaris Resource Manager at any time using the command `srmadm show`. This will output a formatted report of the current operation state of Solaris Resource Manager and its main configuration parameters. This is useful to verify that Solaris Resource Manager is active and all the controlling parameters are active. It also shows the values of global parameters such as the decay rate and location of the Solaris Resource Manager data store.

It is possible to run Solaris Resource Manager without limits active and without CPU scheduling active, which can be useful at startup for debugging and for initially configuring the Solaris Resource Manager product:

```
# srmadm set share=n:limits=n
```

Sharing a Machine Across Departments

A different development group would like to purchase an upgrade for this machine (more processors and memory) in exchange for gaining access to the system when it is idle. Both groups should benefit. To set this up, establish a new group called *development* at the same level as *databases* and *batch*. Allocate *development* 33 percent of the machine since they have added 50 percent more CPU power and memory to the original system.

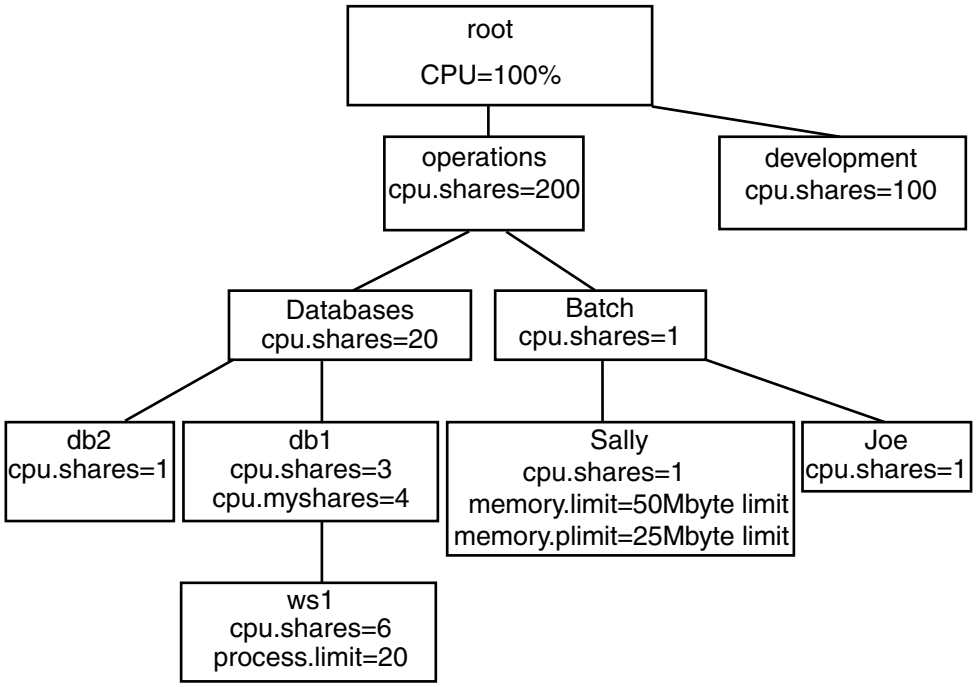


FIGURE 10–5 Sharing a Machine, Step 1

The Development group has hundreds of users. To avoid being involved in the distribution of that group's resources, use the administration flag capability of Solaris Resource Manager to enable the Development system administrator to allocate their resources. You set up limits at the operations and development level as agreed jointly and then you each do the work required to control your own portions of the machine.

To add the new level into the hierarchy, add the group *operations* as a new lnode, and change the parent group of *batch* and *databases* to *operations*:

```
# limadm set sgroup=operations batch databases
```

To set the administration flag:

```
# limadm set flag.admin=set operations development
```

Since under normal circumstances all servers have daemons and backup processes to be run, these should be added on a separate high-level lnode.

Note – Do not use the `root` lnode, since it has no limits.

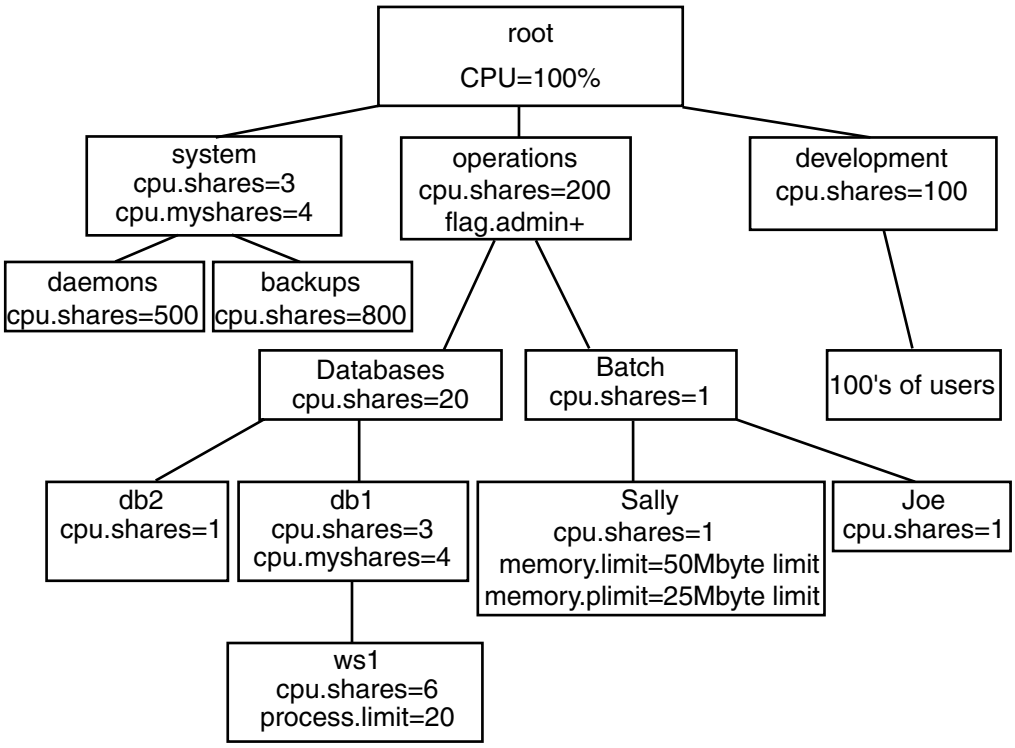


FIGURE 10–6 Sharing a Machine, Step 2

As seen in the examples, you can use Solaris Resource Manager to consolidate several different types of users and applications on the same machine. By the judicious use of CPU share controls, virtual memory limits, process limits, and login controls, you can ensure that these diverse applications receive only the resources that they need. The limits ensure that no application or user is going to adversely impact any other user's or group of users' application. The Solaris Resource Manager product supports simple reporting tools that show users and system administrators exactly what is happening at any given moment, and over the course of time. The report generation capability can be used to show the breakdown of resource usage across applications and groups for capacity planning and billing purposes.

A Typical Application Server

This output would be displayed from a `liminfo` listing of *db1* at the end of the example in the previous section. Typing:

```
# liminfo db1
```

produces:

```
# liminfo  db1
Login name:             db1      Uid(Real,Eff):    223 (223,223)
Sgroup (uid) :    other (98)     Gid(Real,Eff):       50 (50,50)

Shares:                  3       Myshares:                    6
Share:                60.0%      E-share:                  35.4%
Usage:                76000      Accrued usage:          6.4e+08

Mem usage:           11.06 B     Term usage:                 0s
Mem limit:               0 B     Term accrue:                0s
Proc mem limit:          0 B     Term limit:                 0s
Mem accrue:         13.67 TB.s

Processes:               8       Current logins:             1
Process limit:           0

Last used:  Tue Jul 4 15:04:20 1998
Directory:  /usr/people/db1
Name:       Database1
Shell:      /usr/sh/

Flags:  userlimadm+
```

FIGURE 10–7 `liminfo` Listing

The remainder of this section describes the `liminfo` output produced in Figure 9–7. Refer to `liminfo`(1SRM) and `srm`(5SRM) for more information on the fields described below.

- The first two lines of output from the `liminfo` command relate to aspects of the lnode UID and its position in the lnode tree:

Login name
: The login name and initial GID from the password map that corresponds to the UID of the attached lnode. Every lnode is associated with a system UID. A system account should be created for the UID of every lnode. In this instance, a placeholder UID, *db1*, is used for Database1.

Note that the default PAM configuration under Solaris Resource Manager creates an lnode for any user who logs in without one. By default, lnodes created by the superuser or by a user with the `uselimadm` flag set are created with the lnode `srmother` as their parent, or if that does not exist, with the `root` lnode as their parent. The parent of an lnode can be changed with the command generally used to revise lnode attributes, `limadm`.

Uid
The UID of the lnode attached to the current process. Normally, this will be the same as that of the real UID of the process (the logged in user), but in some circumstances (described later) it may differ.

Gid
The GID of the lnode attached to the current process.

R,Euid and R,Egid
The real and effective UID and GID of the current process. This is the same information that is provided by the standard system `id`(1M) command. It is not strictly related to Solaris Resource Manager, but it is displayed for convenience. These fields are not displayed if `liminfo` is displaying information on a user other than the default (that is, if a login name or UID was given as an argument).

Sgroup (`uid`) [`sgroup`]
The name and UID of the parent lnode in the lnode tree hierarchy. This will be blank for the `root` lnode. Many Solaris Resource Manager features depend on the position of an lnode within the tree hierarchy, so it is useful for a user to trace successive parent lnodes back to the root of the tree.

- After the blank line, the next three lines of the `liminfo` display show fields that relate to CPU scheduling:

Shares [`cpu.shares`]
This is the number of shares of CPU entitlement allocated to this user. It is only directly comparable to other users with the same parent lnode, and to the `Myshares` value of the parent lnode itself. Administrators might normally set the shares of all users within a particular scheduling group to the same value (giving those users equal entitlements). This value will normally be something greater than 1, so that administrators have some leeway to decrease the shares of specific users when appropriate.

Myshares [`cpu.myshares`]
This value is only used if this user has child lnodes (that is, if there are other lnodes that have an `sgroup` value of this user) that are active (that is, have processes attached). When this is the case, this value gives the relative share of CPU for processes attached to this lnode, compared with those attached to its child lnodes.

Share
The calculated percentage of the system CPU resources to which the current user is entitled. As other users log in and log out (or lnodes become active or inactive), this value will change, because only active users are included in the calculation. Recent usage by the current user is not included in this calculation.

E-share
This is the effective share of this user (that is, the actual percentage of the system CPU resources that this user would be given in the short term if the user required it and all other active users were also demanding their share). It can be thought of as the current willingness of Solaris Resource Manager to allocate CPU resources to that lnode. This value will change over time as the user uses (or refrains from using) CPU resources. Lnodes that are active but idle (that is, with attached processes sleeping), and so have a low usage, will have a high effective share value. Correspondingly, the effective share can be very small for users with attached processes that are actively using the CPU.

Usage [`cpu.usage`]
The accumulated usage of system resources that are used to determine scheduling priority. Typically, this indicates recent CPU usage, though other parameters may also be taken into account. The parameter mix used can be viewed with the `srmadm` command. Each increment to this value decays exponentially over time so that Solaris Resource Manager will eventually "forget" about the resource usage. The rate of this decay is most easily represented by its half-life, which can be seen with the `srmadm` command.

Accrued usage [`cpu.accrue`]
This is the same resource accumulation measurement as `cpu.usage`, but it is never decayed. It is not used directly by Solaris Resource Manager but can be used by administration for accounting purposes. Unlike usage, this value represents the sum of the accrued usages for all lnodes within the group, as well as that of the current lnode.

- After the second blank line, the next four lines of the `liminfo` listing show fields that relate to virtual memory and terminal usage:

Mem usage [`memory.usage`][`memory.myusage`]
This is the combined memory usage of all processes attached to this lnode.

If two values are displayed, separated by a frontslash (`/`) character, then this lnode is a group header. The first value is the usage for the whole scheduling group, while the second value is that of the current user only.

Mem limit [`memory.limit`]
The maximum memory usage allowed for all processes attached to this lnode and its members (if any). That is, the sum of the memory usage for all processes within the group plus those attached to the group header will not be allowed to exceed this value. Note that in this instance, a value of zero (0) indicates that there is no limit.

Proc mem limit [`memory.plimit`]
The per-process memory limit is the maximum memory usage allowed for any single process attached to this lnode and its members.

Mem accrue [`memory.accrue`]
The `memory.accrue` value is measured in byte-seconds and is an indication of overall memory resources used over a period of time.

Term usage [`terminal.usage`]
The number of seconds of connect-time currently charged to the group.

Term accrue [`terminal.accrue`]
The number of seconds of connect-time used by the group.

Term limit [`terminal.limit`]
The maximum allowed value of the `terminal.usage` attribute. If zero, there is no limit, unless limited by inheritance.

- After the third blank line, the next two lines of the `liminfo` listing display fields that relate to the user and processes:

Processes [`process.usage`][`process.myusage`]
The number of processes attached to this lnode. Note that this refers to processes, not a count of threads within a process.

If two values are displayed, separated by a frontslash (/) character, then this lnode is a group header and the first value is the usage for the whole scheduling group, while the second value is that of just the current user.

Process limit [`process.limit`]
The maximum total number of processes allowed to be attached to this lnode and its members.

Current logins [`logins`]
The current number of simultaneous Solaris Resource Manager login sessions for this user. When a user logs in through any of the standard system login mechanisms (including `login`(1), `rlogin`(1)—basically, anything that uses PAM for authentication and creates a `utmp`(4) entry), this counter is incremented. When the session ends, the count is decremented.

If a user's `flag.onelogin` flag evaluates to set, the user is only permitted to have a single Solaris Resource Manager login session.

- After the fourth blank line, the next four lines of the `liminfo` display show these fields:

Last used [`lastused`]
This field shows the last time the lnode was active. This will normally be the last time the user logged out.

Directory
The user's home directory (items from the password map rather than from Solaris Resource Manager are shown for convenience).

Name
The *db1* (`finger`) information, which is usually the user's name (items from the password map rather than from Solaris Resource Manager are shown for convenience).

Shell
The user's initial login shell (items from the password map rather than from Solaris Resource Manager are shown for convenience).

- After the fifth blank line, the last line of the `liminfo` display shows this field:

Flags
Flags that evaluate to set or group in the lnode are displayed here. Each flag displayed is followed by suffix characters indicating the value and the way in which the flag was set (for example, whether it was explicitly from this lnode (+) or inherited (^)).

Configuring Solaris Resource Manager in Sun Cluster 3.0 Update Environments

Valid Topologies

You can install Solaris Resource Manager on any valid Sun Cluster 3.0 Update topology. See *Sun Cluster 3.0 12/01 Concepts* for descriptions of valid topologies.

Determining Requirements

Before you configure the Solaris Resource Manager product in a Sun Cluster environment, you must decide how you want to control and track resources across switchovers or failovers. If you configure all cluster nodes identically, usage limits will be enforced identically on primary and backup nodes.

While the configuration parameters need not be identical for all applications in the configuration files on all nodes, all applications must at least be represented in the configuration files on all potential masters of that application. For example, if Application 1 is mastered by *phys-schost-1* but could potentially be switched or failed-over to *phys-schost-2* or *phys-schost-3*, then Application 1 must be included in the configuration files on all three nodes (*phys-schost-1*, *phys-schost-2*, and *phys-schost-3*).

Solaris Resource Manager is very flexible with regard to configuration of usage and accrual parameters, and few restrictions are imposed by Sun Cluster. Configuration choices depend on the needs of the site. Consider the general guidelines in the following sections before configuring your systems.

Configuring Memory Limits Parameters

When using the Solaris Resource Manager product with Sun Cluster, you should configure memory limits appropriately to prevent unnecessary failover of applications and a ping-pong effect of applications. In general:

- Do not set memory limits too low.

  When an application reaches its memory limit, it might fail over. This is especially important for database applications, when reaching a virtual memory limit can have unexpected consequences.

- Do not set memory limits identically on primary and backup nodes.

  Identical limits can cause a ping-pong effect when an application hits its memory limit and fails over to a backup node with an identical memory limit. Set the memory limit slightly higher on the backup node. The applications, resources, and preferences at the site determine how much higher the limit is set. The difference in memory limits helps prevent the ping-pong scenario and gives you a period of time in which to adjust the parameters as necessary.

- Do use the Solaris Resource Manager memory limits for coarse-grained problem scenario load-balancing.

  For example, you can use memory limits to prevent an errant application from consuming excess resources.

Using Accrued Usage Parameters

Several Solaris Resource Manager parameters are used for keeping track of system resource usage accrual: CPU shares, number of logins, and connect-time. However, in the case of switchover or failover, usage accrual data (CPU usage, number of logins, and connect-time) will restart at zero by default on the new master for all applications that were switched or failed over. Accrual data is not transferred dynamically across nodes.

To avoid invalidating the accuracy of the Solaris Resource Manager usage accrual reporting feature, you can create scripts to gather accrual information from the cluster nodes. Because an application might run on any of its potential masters during an accrual period, the scripts should gather accrual information from all possible masters of a given application. For more information, see Chapter 9.

Failover Scenarios

On Sun Cluster, Solaris Resource Manager can be configured so that the resource allocation configuration described in the lnode configuration (`/var/srm/srmDB`) remains the same in normal cluster operation and in switchover or failover situations. For more information, see “Sample Share Allocation” on page 65.

The following sections are example scenarios.

- The first two sections, “Two-Node Cluster With Two Applications” on page 115 and “Two-Node Cluster With Three Applications” on page 116, show failover scenarios for entire nodes.
- The section “Failover of Resource Group Only” on page 118 illustrates failover operation for an application only.

In a cluster environment, an application is configured as part of a resource group (RG). When a failure occurs, the resource group, along with its associated applications, fails over to another node. In the following examples, Application 1 (App-1) is configured in resource group RG-1, Application 2 (App-2) is configured in resource group RG-2, and Application 3 (App-3) is configured in resource group RG-3.

Although the numbers of assigned shares remain the same, the percentage of CPU resources allocated to each application will change after failover, depending on the number of applications running on the node and the number of shares assigned to each active application.

In these scenarios, assume the following configurations.

- All applications are configured under a common parent lnode.
- The applications are the only active processes on the nodes.
- The limits databases are configured the same on each node of the cluster.

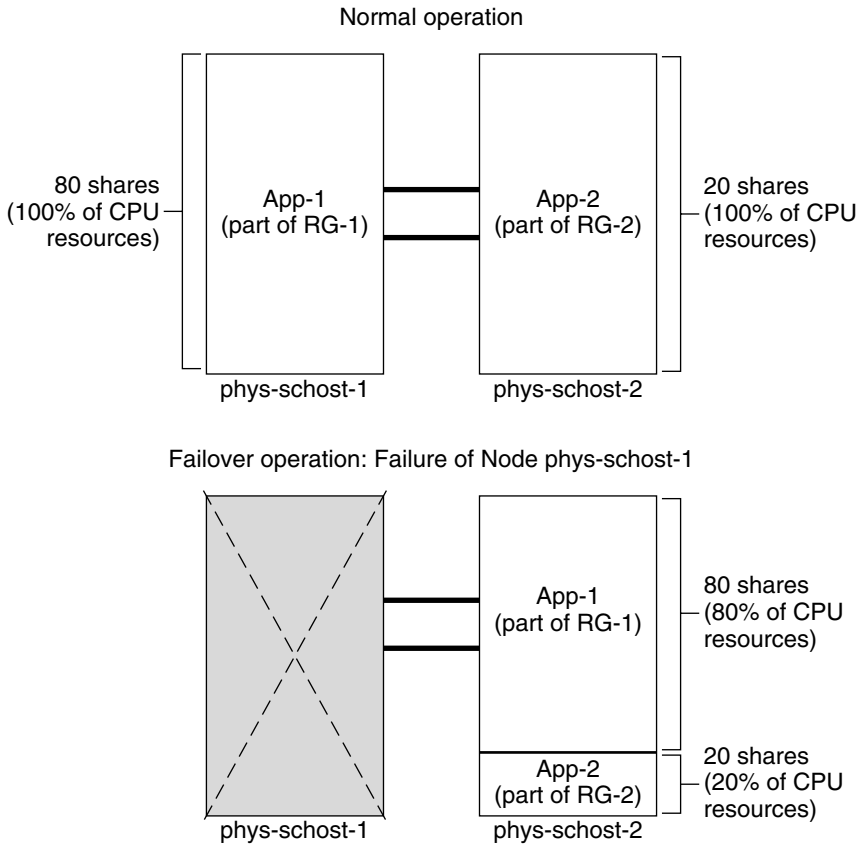
Two-Node Cluster With Two Applications

You can configure two applications on a two-node cluster such that each physical host (*phys-schost-1*, *phys-schost-2*) acts as the default master for one application. Each physical host acts as the backup node for the other physical host. All applications must be represented in the Solaris Resource Manager limits database files on both nodes. When the cluster is running normally, each application is running on its default master, where it is allocated all CPU resources by Solaris Resource Manager.

After a failover or switchover occurs, both applications run on a single node where they are allocated shares as specified in the configuration file. For example, this configuration file specifies that Application 1 is allocated 80 shares and Application 2 is allocated 20 shares.

```
# limadm set cpu.shares=80 App-1
# limadm set cpu.shares=20 App-2
...
```

The following diagram illustrates the normal and failover operations of this configuration. Note that although the number of shares assigned does not change, the percentage of CPU resources available to each application can change, depending on the number of shares assigned to each process demanding CPU time.



Two-Node Cluster With Three Applications

On a two-node cluster with three applications, you can configure it such that one physical host (*phys-schost-1*) is the default master of one application and the second physical host (*phys-schost-2*) is the default master for the remaining two applications. Assume the following example limits database file on every node. The limits database file does not change when a failover or switchover occurs.

```
# limadm set cpu.shares=50     App-1
# limadm set cpu.shares=30     App-2
# limadm set cpu.shares=20     App-3
...
```

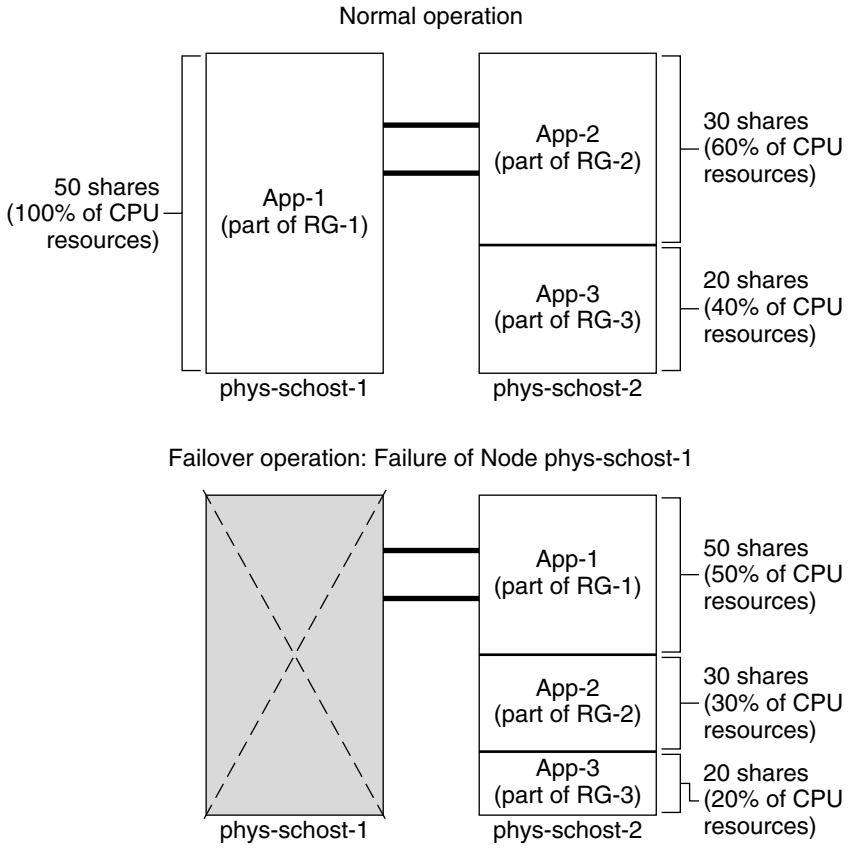
When the cluster is running normally, Application 1 is allocated 50 shares on its default master, *phys-schost-1*. This is equivalent to 100 percent of CPU resources because it is the only application demanding CPU resources on that node.

Applications 2 and 3 are allocated 30 and 20 shares, respectively, on their default master, *phys-schost-2*. Application 2 would receive 60 percent and Application 3 would receive 40 percent of CPU resources during normal operation.

If a failover or switchover occurs and Application 1 is switched over to *phys-schost-2*, the shares for all three applications remain the same, but the percentages of CPU resources are re-allocated according to the limits database file.

- Application 1, with 50 shares, receives 50 percent of CPU.
- Application 2, with 30 shares, receives 30 percent of CPU.
- Application 3, with 20 shares, receives 20 percent of CPU.

The following diagram illustrates the normal and failover operations of this configuration.



Failover of Resource Group Only

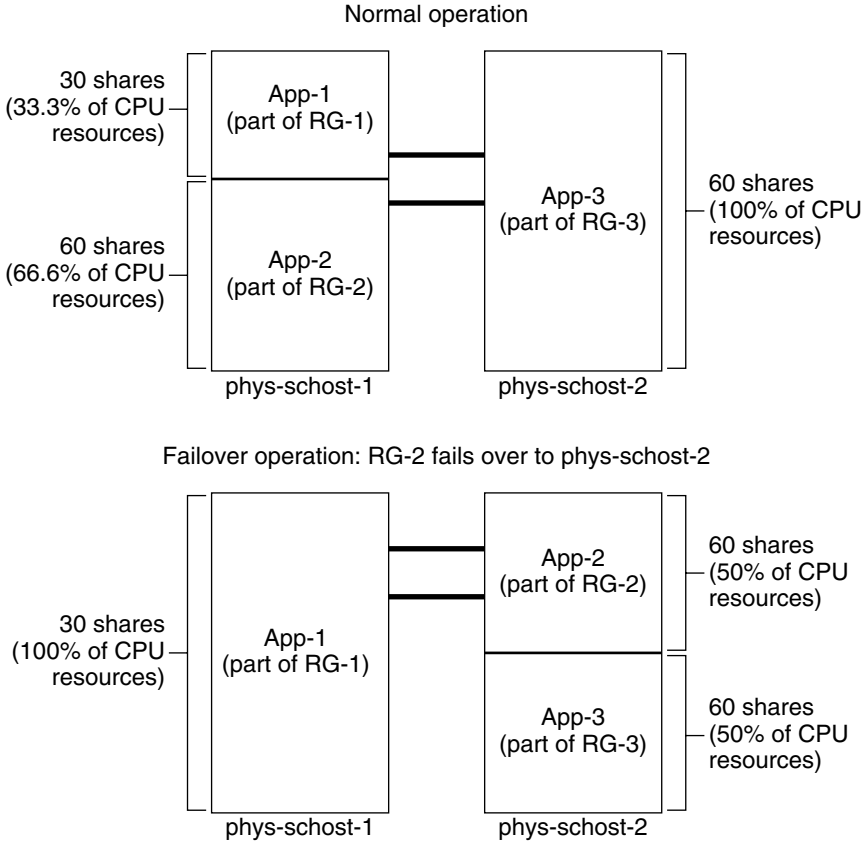
In a configuration in which multiple resource groups have the same default master, it is possible for a resource group (and its associated applications) to fail over or be switched over to a backup node, while the default master remains up and running in the cluster.

Note – During failover, the application that fails over will be allocated resources as specified in the configuration file on the backup node. In this example, the limits database files on the primary and backup nodes have the same configurations.

For example, this sample configuration file specifies that Application 1 is allocated 30 shares, Application 2 is allocated 60 shares, and Application 3 is allocated 60 shares.

```
# limadm set cpu.shares=30 App-1
# limadm set cpu.shares=60 App-2
# limadm set cpu.shares=60 App-3
...
```

The following diagram illustrates the normal and failover operations of this configuration, where RG-2, containing Application 2, fails over to *phys-schost-2*. Note that although the number of shares assigned does not change, the percentage of CPU resources available to each application can change, depending on the number of shares assigned to each application demanding CPU time.



CHAPTER 11

Troubleshooting

This chapter provides pointers for diagnosing problems in the operation of Solaris Resource Manager.

If you require additional assistance, contact your Sun Software Support Provider.

User-Related Problems

User Cannot Log In

- The user has the `nologin` or `noattach` flag set.
- The user has the `onelogin` flag set and is already logged in at another terminal or window.
- The user has reached the connect-time usage limit. The user needs to wait for the usage to decay before logging in again, or the administrator can change either the user's `terminal.usage` attribute or `terminal.limit` attribute to give the user additional terminal connect-time.
- The user's lnode may exist but it has been orphaned because its parent lnode has been removed. See "Orphaned Lnodes" on page 126.

None of the Solaris Resource Manager limitations listed above apply to the superuser.

Note – Although the user is able to log in to the system, if there is no lnode corresponding to the UID of the user (an lnode has not been set up for that user's account), the problem is identified by a message indicating that: No limits information is available. Refer to "Orphaned Lnodes" on page 126.

User Not Informed of Reaching Limits

During normal operation of Solaris Resource Manager, a logged-in user receives notification messages whenever a limit is reached. Sometimes users miss seeing these communications and are unaware of the cause of any problems they are having, and the system will appear to behave mysteriously. However, the system administrator will have been notified.

The delivery of notification messages is carried out by the Solaris Resource Manager daemon program, `limdaemon`. There are a number of possibilities that the administrator can investigate if notification messages are not being delivered to users:

- The console window is hidden. If a user has logged in using a particular window and then opened additional windows that cover the login window, the user may miss a message delivered to the login window.
- The `limdaemon` program is not running.
- `limdaemon` is unable to dynamically allocate additional memory to maintain its internal structures. If this happens, `limdaemon` displays a diagnostic message on the system console the first time that it fails to get sufficient memory. It continues to attempt to get memory, but fails silently after the first attempt.
- The `utmp` file is corrupt or missing. `limdaemon` relies on this file to identify the terminals where a user is logged in, so notification messages can be sent to those terminals. If the `utmp` file is corrupted or missing, an error message is reported on the console, and the notification message delivery is suppressed.
- `limdaemon` is unable to deliver a message due to a system limitation. For example, if `limdaemon` needs to open a window on a terminal to deliver the message and is unable to, then the message is dropped.

Unable to Change User's Group

The `sgroup` attribute determines the lnode's parent in the scheduling tree. This hierarchy is used to regulate resource usage and to schedule the CPU. For this reason several security precautions are placed on modifying the `sgroup` attribute, both to avoid inadvertent errors when changing it and to prevent circumvention of Solaris Resource Manager.

To modify the `sgroup` attribute, a user needs one of the following privileges:

- Be the superuser
- Have a set `uselimadm` flag
- Have a set `admin` flag and be a group header for the lnode being changed

Orphaned lnodes cannot be made parents of other lnodes. See "Orphaned Lnodes" on page 126.

Users Frequently Exceeding Limits

Check whether any of these conditions are causing the problem:

- A user's administrative limit is set too low for the user's requirements.
- The usage attribute is not being decayed. The administrator is responsible for ensuring that decays are performed on the device categories for all renewable resources (including the terminal device category). Typically, this would be done by regular execution of the `limdaemon` command. If a decay is not performed for a renewable resource, the usage attribute for that resource continues to increase until its limit is reached.
- The decay period is too long. The frequency of execution of `limdaemon` should be set to accommodate the granularity of the shortest decay interval.
- The decay attribute for a renewable resource is too small, or the interval attribute is too large. If the decay for a renewable resource over a given time interval is set below the typical consumption rate of that resource, the usage attribute will gradually increase until its limit is reached.

Unexpected Notification Messages

Notification messages will be received by any user affected by a limit being reached. Therefore, if a group limit is reached, the group header, and all users below in the scheduling hierarchy, will receive a notification message.

If a user is attached to another lnode and a limit is reached, the user will not receive a notification message, but all the other affected users will. The cause of the problem may not be apparent to the group that is affected.

Terminal Connect-Time Not Updated

The most likely cause of this problem is that the `limdaemon` program is not running. `limdaemon` periodically updates the usage and accrue attributes in the terminal device category for all currently logged in users. Typically, it would be started from the Solaris Resource Manager `init.d` script.

Performance Issues

Processes Attached to the `root` Lnode

For reasons of system management, processes attached to the `root` lnode are given almost all the CPU resources they demand. Therefore, if a CPU-bound process is attached to the `root` lnode, it will tie up a CPU, causing processes on other lnodes to slow or stop.

The following precautions can be taken to prevent this from occurring:

- The administrator should always log in to an lnode created for normal use by the administrator, rather than attaching to the `root` lnode. If there is a need to attach to the `root` lnode, be careful not to use any CPU-intensive applications, such as compilers. To use a UID of superuser without attaching to the `root` lnode, the administrator can use the `su`(1) command.
- The `init.d` scripts can be changed to use the `srmuser` program to attach all daemons to lnodes of their own, so they are not attached (by inheritance) to the `root` lnode. However, this solution cannot be routinely recommended. It can be a burden since a large number of files need to be edited, and the practice could inhibit the ability to integrate patches into a system later. A resolution that does not require this task to be performed manually is under investigation.

  For Solaris Resource Manager releases after 1.0, the scripts `sbin_rc2` and `sbin_rc3` provided in the `/usr/srm/unsupport` directory can be used to partially solve this problem.

A program that runs as `setuid-root` does not automatically attach to the root lnode. Normally, the process remains attached to the lnode of the parent that created it, and only the effective UID is changed.

CPU Resources Not Controlled by Solaris Resource Manager

Solaris Resource Manager only controls CPU use by processes in the SHR scheduling class. If excessive demands are made at higher priority by other scheduling classes, especially real-time (RT) and system (SYS), then SHR can only schedule with the residual CPU resource.

The use of the RT class conflicts with the Solaris Resource Manager software's ability to control the system. Real-time processes get complete access to the system, specifically so they can deliver real-time response (generally on the order of a few hundred microseconds). Processes running in the SHR class by definition have lower priority than anything running in real time, and Solaris Resource Manager has no control over RT processes. Real-time processes can easily consume *all* available resources, leaving Solaris Resource Manager nothing left to allocate to remaining processes.

One notable system service that runs *entirely* in the SYS class is the NFS server. Solaris Resource Manager cannot control the NFS daemons, because they run in SYS. The Solaris Resource Manager product's ability to allocate processor resources may be reduced on systems offering extensive NFS service.

While processes are executing kernel code (inside a system call), the usual time-slice preemption rules do not apply. Most system calls will only do a reasonable amount of work before they reach a preemption point. However, if the system is under high load from more intensive system calls, this can result in reduced overall responsiveness and is outside the control of a scheduling class.

If the system is short of available real memory, then the resulting I/O bottleneck as the page fault rate increases and process swapping increases leads to increased kernel consumption of CPU. Large amounts of time spent waiting on I/O may indicate lost CPU capacity. Again, this is outside the scope of a scheduling class to control.

The SHR scheduling class is a time-sharing (TS) scheduler. It uses the same global priority range as the TS and the interactive (IA) schedulers. It is not appropriate to mix the use of SHR with TS and IA except during the transition of moving all processes into or out of the SHR class. System operation with a mix of processes in SHR and TS classes will result in reduced quality of scheduling behavior in both classes. For this reason, Solaris Resource Manager prevents non-`root` processes from moving themselves or others to the TS or IA classes. The RT class uses an alternate priority range and may be used with the SHR class in the same way as the TS and IA classes.

If processes run by superuser contain code that uses the `priocntl`(2) system call directly instead of using the `setpriority`(3C) library routine to adjust process priorities, then the target processes may be moved into another scheduling class

(typically TS). The `setpriority` library routine code accounts for the fact that the `priocntl` interface to SHR is binary compatible with that of TS and thus avoids the problem. The `-c` option of `ps`(1) or the `-d` option of `priocntl`(1) can be used to display the scheduling class of processes.

The same difficulty arises with superuser privilege processes that explicitly use `priocntl`(2) to manage the scheduling class membership of processes.

Orphaned Lnodes

An *orphaned* lnode is one that has a nonexistent parent lnode. This is of concern to an administrator because Solaris Resource Manager prevents processes from attaching to any lnode that is orphaned or has an orphaned ancestor in the scheduling tree.

The kernel checks changes to the `sgroup` attribute in order to prevent the creation of orphans by invalid alterations to the scheduling group parent.

The major effect of an lnode being orphaned is that it can no longer have processes attached to it. Since no process can connect to it, the lnode cannot be used for logging in. Any attempts to log in using the corresponding account will fail.

The easiest way for an administrator to detect orphaned lnodes is to use the `limreport` command with the built-in orphan identifier. The command:

```
% limreport orphan - uid sgroup lname
```

will list the UID, scheduling group parent, and login name of users who have orphaned lnodes. The `sgroup` attribute can be used to determine which of the lnodes is at the top of an orphaned section of the tree.

The first step an administrator should take when an orphaned lnode is discovered is to find the top of the orphaned section of the scheduling tree, since this is the lnode that needs to be reattached. If the top of the orphaned section is not correctly identified, only part of the orphaned section will be reattached to the tree.

When the top of the orphaned section has been determined, an administrator with sufficient privilege can use `limadm` to set the `sgroup` attribute of the topmost orphaned lnode to a valid lnode within the scheduling tree. This will cause the orphaned lnode to be reattached to the tree as a member of the group that the valid lnode heads. `limadm` verifies that the new scheduling group parent to be applied is able to be activated, thus ensuring that the lnode being changed will no longer be orphaned.

Alternatively, the administrator can create a new user whose UID is equal to the UID in the `sgroup` attribute of the orphaned lnode. This will cause the automatic reattachment of the orphaned section of the tree.

Group Loops

When an lnode is made active, all of its parents up to the `root` lnode are also activated. As a result of this process, if one of the lnodes is seen to have a parent that has already been encountered, the kernel has discovered a *group loop*.

If the limits database is corrupted, it is possible for a group loop to occur, in which one of the *ancestors* of an lnode is also one of its children. When the kernel discovers a group loop, it silently and automatically connects the loop into the scheduling tree by breaking it arbitrarily and connecting it as a group beneath the `root` lnode. The kernel cannot determine which is the uppermost lnode since the loop has no beginning or end. This means that the lnode at the point where the loop is connected to the scheduling tree becomes a group header of a topmost group. It is possible that members of this group might inherit privileges or higher limits than they would otherwise have.

Cause

Group loops are prevented by `limadm` when setting scheduling group parents. A group loop can only occur through corruption to the limits database. This is a serious problem, and may cause all sorts of other difficulties in Solaris Resource Manager since the limits database is so basic to its operation.

Correction

The problem is self-correcting with respect to the structure of the scheduling tree since the kernel attaches the lnode to the `root` lnode. Because the attachment is from an arbitrary point in the loop, the administrator has to determine where the lnode should be attached and also check the point of attachment for every other member in the loop.

The result of automatic group loop repair can be seen by listing the lnodes that are children of the `root` lnode. The command:

```
% limreport 'sgroup==0' - uid lname
```

will list all lnodes that have the `root` lnode as their parent. If any lnodes are listed that should not be children of the `root` lnode, they are possibly the top of a group loop that has been attached beneath the `root` lnode.

The major concern for the administrator when a group loop is detected is that, since the cause of the group loop was corruption to the limits database, many more serious problems could arise. If the administrator suspects corruption in the limits database, it is best to carry out some validation checks against the file to determine if it has been corrupted and then take remedial action. Refer to "Crash Recovery" on page 128 for details on detecting and correcting a corrupt limits database.

Resolving UID Conflicts

To verify that the UIDs assigned by the system for `srmidle`, `srmlost`, and `srmother` are not in conflict with any existing UIDs, type:

```
# /usr/bin/egrep 41\|42\|43 /etc/passwd
```

If a conflict exists, you can change the UIDs by editing the password and shadow files, `/etc/passwd` and `/etc/shadow`.

Crash Recovery

There are many concerns for an administrator when a Solaris system has a failure, but there are some additional considerations when a Solaris Resource Manager system is being used. They are:

- The limits database may have been corrupted by a disk error or for some other reason
- The operation of `limdaemon` during a system failure, particularly in relation to connect-time usage charging, may be faulty

The following sections discuss these in detail and offer suggestions for handling the situation, where appropriate.

Corruption of the Limits Database

Solaris Resource Manager maintenance of the limits database is robust and corruption is unlikely. However, if corruption does occur, it is of major concern since this database is basic to the operation of Solaris Resource Manager. Any potential corruption should be investigated and, if detected, corrected.

Symptoms

No single symptom can reliably be used to determine whether the limits database has been corrupted, but there are a number of indicators that potentially reflect a corrupted limits database:

- The detection of a group loop by the Solaris Resource Manager kernel is a positive indication that the limits database has been corrupted. Group loops are strictly prevented by Solaris Resource Manager, and they can only occur when an `sgroup` attribute has become corrupted in some way. Refer to "Group Loops" on page 127 for more details.
- Users seeing the message 'No limits information available' displayed when they attempt to log in, and their logins being rejected. This can occur if the corruption to the limits database causes their `flag.real` attributes to be cleared, which effectively deletes their lnodes. It will affect not only the lnode which is deleted, but also any lnodes which are orphaned (refer to "Orphaned Lnodes" on page 126 for details). Note that the 'No limits information available' message will also appear if no lnode has been created for the account or if the lnode has been intentionally deleted, so it is not a clear indicator that the limits database has been corrupted.
- Unrealistic values suddenly appearing in usage or limits attributes. This may cause some users to suddenly hit limits.
- Users suddenly complaining of a loss of privilege or unexpected privileges, caused by corruption of privilege flags.

If an administrator suspects that there is corruption in the limits database, the best way to detect it is to use `limreport` to request a list of lnodes with attributes that should have values within a known range. If values outside that range are reported, corruption has taken place. `limreport` could also be used to list lnodes which have a clear `flag.real`. This will indicate accounts in the password map for which no lnode exists.

Correction

When corruption is detected, the administrator should revert to an uncorrupted version of the limits database. If the corruption is limited to a small section of the limits database, the administrator may be able to save the contents of all other lnodes and reload them into a fresh limits database using the `limreport` and `limadm`

commands. This would be preferable if no recent copy of the limits database is available since the new limits database would now contain the most recent usage and accrue attributes. The procedure for saving and restoring the limits database is documented in Chapter 5. For simple cases of missing lnodes it could be sufficient to just recreate them by using the `limadm` command.

Connect-Time Loss by `limdaemon`

If `limdaemon` terminates for any reason, all users currently logged in cease to be charged for any connect-time usage. Furthermore, when `limdaemon` is restarted, any users logged in will continue to use those terminals free of charge. This is because the daemon relies on login notifications from `login` to establish a Solaris Resource Manager login session record within the internal structures it uses to calculate connect-time usages. Therefore, whenever it starts, there are no Solaris Resource Manager login sessions established until the first notification is received.

Typically this will not be a problem if `limdaemon` terminated due to a system crash, since the crash will also cause other processes to terminate. Login sessions would then not be able to recommence until the system is restarted.

If `limdaemon` terminates for some other reason, the administrator has two choices:

1. Restart the daemon immediately, and ignore the lost charging of terminal connect-time for users who are already logged in. This could mean that a user has free use of a terminal indefinitely unless identified and logged out.
2. Bring the system back to single-user mode then return to multi-user mode, thus ensuring that all current login sessions are terminated and users can only log in again after the daemon has been restarted.

CHAPTER 12

Notification Messages

The Solaris Resource Manager system error messages are:

`memory limit reached`
The virtual memory usage (`memory.usage`) of an lnode has reached the virtual memory limit (`memory.limit`).

`process limit reached`
The number of processes (`process.usage`) of an lnode has reached the process count limit (`process.limit`).

`per-process memory limit reached`
The virtual memory usage of a process has reached the virtual memory limit (`memory.plimit`).

`lnode attach failed in setuid`
If the Solaris Resource Manager software is installed and enabled, then the `setuid` system call, in addition to its standard function, attaches the calling process to the lnode associated with its new real UID. If attachment fails, it is usually because there is no lnode associated with the new UID.

`currently barred from logging in`
The `flag.nologin` is set for the user at the time the user tries to log in.

`already logged in - only one login allowed`
The `flag.onelogin` is set for the user, and the user has already logged in from another terminal.

`no permission to use this terminal`
The `terminal.flag.all`, `terminal.flag.console`, `terminal.hardwired`, or `terminal.flag.network` is set for the user when the user tries to log in from a particular terminal.

`terminal connect-time limit reached`
The terminal connect-time (`terminal.usage`) has reached the limit (`terminal.limit`).

APPENDIX **A**

Solaris Resource Manager Code Examples

Initialization Script

The following startup script is supplied with the system as `/etc/init.d/init.srm`. It is executed with an argument of `start` as the system is changing to run-level 2 or 3 (multi-user mode). It is also executed with an argument of `stop` at system shutdown.

```
#!/bin/sh
#
# Copyright (c) 1998-1999 by Sun Microsystems, Inc.
# All rights reserved.
#
# Copyright 1995-1997 Softway Pty. Ltd.
#
# Start/stop Solaris Resource Manager v1.1
#
#ident  "@(#)init.srm 1.24 99/02/10 SMI"

#############################################################################
# Default values.

DATADIR=/var/srm
ShareDb=$DATADIR/srmDB
LimdaemonOptions=
ChargeOptionsOn="limits=y:share=y:adjgroups=y:limshare=y"
ChargeOptionsOff="limits=n:share=n:adjgroups=n:limshare=n"
DaemonLnode=daemon
LostLnode=srmlost
IdleLnode=srmidle
OtherLnode=srmother

#############################################################################

# ECHO=echo    # For a verbose startup and shutdown
```

```
ECHO=:        # For a quiet startup and shutdown

SRMDIR=/usr/srm
SRMBIN=$SRMDIR/bin
SRMSBIN=$SRMDIR/sbin
SRMLIB=$SRMDIR/lib
ETCSRM=/etc/srm

PATH=/sbin:/usr/sbin:/bin:$PATH:$SRMSBIN:$SRMBIN:$SRMLIB
export PATH
case "$1" in
'start')
    if [ ! -x $SRMSBIN/srmadm ]; then
        echo "Solaris Resource Manager *not* installed." \
            "Missing srmadm command."
        exit
    fi

    # Only bother if sched/SHR is loaded.
    if [ `$SRMSBIN/srmadm` != yes ]
    then
        #
        # Usually this is because /etc/system doesn't have the usual
        #    set initclass="SHR"
        # or at least a set extraclass="SHR"
        #
        echo "Solaris Resource Manager *not* loaded."
        exit
    else
        echo "Enabling Solaris Resource Manager"
        if [ `$SRMSBIN/srmadm show fileopen` = yes ]; then
            echo "SRM database file already open - stopping first."
            limdaemon -k
            sleep 2
            srmadm set $ChargeOptionsOff
            sync
            srmadm set fileopen=n
            $ECHO "SRM inactive"
        fi
        $ECHO "Starting SRM..."
    fi

    # Check the limconf file.
    if [ ! -s $ETCSRM/limconf ]; then
        echo "SRM - file $ETCSRM/limconf is missing " >&2
        echo "SRM not started."
        exit 1
    fi

    if [ ! -f "$ShareDb" ]; then
        echo "SRM database '$ShareDb' not present - " \
            "creating empty database"
        if [ ! -d "$DATADIR" ]; then
            mkdir "$DATADIR"
            chmod 400 "$DATADIR"
```

```
        chown root "$DATADIR"
        chgrp root "$DATADIR"
    fi
    touch "$ShareDb" ||
    {
        echo "Failed to create '$ShareDb'" >&2
        echo "SRM not started"
        exit 1
    }
    chmod 400 "$ShareDb"
    chown root "$ShareDb"
    chgrp root "$ShareDb"
fi

CreateLnodes=0
if [ ! -s "$ShareDb" ]; then
    $ECHO "SRM Warning: Using empty database" >&2
    CreateLnodes=1
fi

$ECHO "SRM starting ... \c"

# Open Lnode file.
srmadm set -f "$ShareDb" fileopen=y

if [ $? != 0 ]; then
    echo
    echo "srmadm set -f $ShareDb failed" >&2
    echo "SRM not started"
    exit 1
fi

# Set SRM global options.
srmadm set $ChargeOptionsOn
if [ $? != 0 ]; then
    echo
    echo "srmadm set $ChargeOptionsOn failed" >&2
    echo "SRM not completely enabled"
    exit 1
fi

# Create if needed the daemon lnode.
liminfo "$DaemonLnode" 2>/dev/null | \
    grep "^Login name:  *$DaemonLnode " >/dev/null 2>&1
if [ $? -ne 0 ]; then
    # If daemon lnode does not, create one.
    limadm set cpu.shares=1 "$DaemonLnode" 2>/dev/null
    limadm set sgroup=root "$DaemonLnode" 2>/dev/null
fi

# Create if needed the other lnode.
liminfo "$OtherLnode" 2>/dev/null | \
    grep "^Login name:  *$OtherLnode " >/dev/null 2>&1
if [ $? -ne 0 ]; then
    # If "other" sgroup exists but has no lnode, create one.
```

```
    limadm set cpu.shares=1 "$OtherLnode" 2>/dev/null
    limadm set sgroup=root "$OtherLnode" 2>/dev/null
fi

# Create if needed, and set the lost lnode.
if [ x"$LostLnode" != x ]; then
        liminfo "$LostLnode" 2>/dev/null | \
        grep "^Login name:  *$LostLnode " >/dev/null 2>&1
    if [ $? -ne 0 ]; then
        limadm set cpu.shares=1 "$LostLnode"
        limadm set sgroup=root "$LostLnode"
    fi

    srmadm set lost="$LostLnode" ||

    $ECHO "SRM - Warning: No user '$LostLnode' for lost lnode"
fi

# Create if needed, and set the idle lnode.
if [ x"$IdleLnode" != x ]; then
        liminfo "$IdleLnode" 2>/dev/null | \
        grep "^Login name:  *$IdleLnode " >/dev/null 2>&1
    if [ $? -ne 0 ]; then
        limadm set cpu.shares=0 "$IdleLnode"
        limadm set sgroup=root "$IdleLnode"
    fi
    srmadm set idle="$IdleLnode" ||
    $ECHO "SRM - Warning: No user '$IdleLnode' for idle lnode"
fi

# If creating SRM database, set up existing users.
if [ "$CreateLnodes" -eq 1 ]; then
    echo "SRM - creating user lnodes; may take a while"
# We now want to catch any other users which were not found
# on the filesystems.  First we need to decide what the maximum
# uid value we will create an l-node entry for in the database.
# We choose less than the uid for 'nobody' so that we can try
# and minimise the apparent size of the database (which is a sparse
# file).  If the user 'nobody' does not exist then we just have
# to take our chances with using all possible uid values.
# Unfortunately all this means that there are certain circumstances
# where not all users will be taken into account.

MaxUID=`awk -F: "\\$1==\"nobody\" { print \\$3 - 100 }" /etc/passwd`
    if [ $? -eq 0 -a x"$MaxUID" != x ] ; then
        Cond="uid >= 0 && uid < $MaxUID && !flag.real"
    else
        Cond="uid >= 0 && !flag.real"
    fi
    UIDS=`limreport "$Cond" '%d\n' uid | wc -l`
    if [ $UIDS -gt 0 ]; then
        $ECHO "$UIDS other lnodes to be created" \
            "due to passwd entries"
        CMDS="limadm set cpu.shares=1:sgroup=$OtherLnode"
        limreport "$Cond" "$CMDS %d\necho ' uid %7d\r\c'\n" \
```

```
                uid uid | sh
            echo
        fi
    fi

    limdaemon $LimdaemonOptions

    echo "Solaris Resource Manager Enabled."
    ;;

'stop')
    # SRM shutdown should be done as late as possible before
    # filesystems are unmounted.
    if [ -x $SRMSBIN/srmadm ] && $SRMSBIN/srmadm show fileopen > /dev/null
    then
        limdaemon -k
        sleep 2
        srmadm set $ChargeOptionsOff
        srmadm set fileopen=n
        sync
        $ECHO "Solaris Resource Manager Disabled"
    fi
    ;;

*)
    echo "Usage: $0 {start|stop}"
    ;;
esac
```

Default 'no lnode' Script

This script creates the lnode in the default scheduling group (`other` if such a user exists in the password map, otherwise `root`) and mails the system administrator a reminder to move the new lnode into the appropriate place in the scheduling hierarchy.

```
#!/bin/sh
#
#ident  "@(#)nolnode.sh 1.10 99/05/07 SMI"
#
# Copyright (c) 1998-1999 by Sun Microsystems, Inc.
# All rights reserved.
#
# Copyright 1995-1997 Softway Pty. Ltd.
#
# A script that called by the PAM module to create a lnode
#

PATH=/usr/srm/sbin:/usr/srm/bin:/sbin:/bin export PATH
LOCALE=C export LOCALE
if [ "$DEBUG" = "true" ]
then
```

```
    exec >> /tmp/nolnodelog 2>&1
    echo
    date
    echo "Attempting to create lnode for $USER"
else
    exec > /dev/null  2>&1
fi

err=`limadm set -u cpu.shares=1 "$UID" 2>&1`
if [ $? -eq 0 ]
then
    SRM_GROUP=`liminfo -v $USER | grep '^sgroupname' | awk '{ print $2 }'`
    SRM_SHARE=`liminfo -v $USER | grep '^cpu.shares' | awk '{ print $2 }'`
    export SRM_GROUP SRM_SHARE
    cat <<-EOF | /usr/lib/sendmail root
    Subject: New lnode created for "$USER"

    Remember to change scheduling group and shares for
    "$USER". Currently in group "$SRM_GROUP" with $SRM_SHARE share.

EOF
else
    cat <<-EOF | /usr/lib/sendmail root
    Subject: Could not create lnode for "$USER"

    after "$SERVICE" attempt on tty "$TTY", uid "$UID",
    rhost "$RHOST",
    limadm said "$err"
EOF
    exit 1 # deny access
fi
# permit access
exit 0
```

Glossary

A number of concepts are introduced in the Solaris Resource Manager product, and there are areas that overlap conceptually with other parts of the Solaris environment.

To simplify the discussion within this document and to avoid confusion, the following terms are defined.

accrue For both fixed and renewable resources there may be an accrue usage attribute, which is the integral of the corresponding usage attribute over time.

active An lnode is active if there are any processes attached to it or attached to any of its descendants. An lnode cannot be removed while it is active.

`admin` **user** A user whose lnode `flag.admin` evalutes to `set`. `admin` users may modify lnodes within their scheduling groups. Also see *group administrator*.

administrator Anyone whose role includes maintaining the system. Solaris Resource Manager provides functionality to allow administrative authority to be delegated without the need to provide superuser privileges. Also see `admin` user, *superuser*, `uselimadm` *user*, *group administrator*, and "Delegated Administration" on page 51.

allocated share The fraction of available CPU resources that would be given to a user in the long term with a given configuration of lnode tree hierarchy, shares, and active lnodes.

ancestor An lnode is defined as the ancestor of another if successive references to the `sgroup` attribute, starting from the first lnode, eventually reference the other. That is, the latter lnode is a descendant, or a member, of the first.

attached When a user logs in, the PAM module process of the user attaches to the lnode corresponding to the user's UID. Any processes which are

subsequently spawned are attached to the same lnode by default. The lnode that a process is attached to determines the process's limits, CPU entitlement, and privileges.

attributes — The data fields of an lnode. Because all lnodes have the same internal structure, all users have the same set of attributes. Attributes may be system, user (only used by user-mode programs), or domain. The different types of attributes differ in the field numbers allotted to them. System attributes are those which are used directly by the kernel, such as the numeric variables to control resources such as processes, memory size, and flags that control system privileges at the kernel level. User attributes can be added at any time by the administrator, and existing user attributes can be modified at any time provided the changes do not disrupt programs that use the attribute. Domain attributes are not declared in the configuration source file, because the declaration of a domain defines them implicitly.

batch workload — A hybrid between online transaction processing (OLTP) and decision support system (DSS) workloads. A batch workload can consist of many repetitive transactions to a database, with some heavy computational work for each.

cap — A limit that is placed on system resource usage.

capping — The process of placing a limit on system resource usage.

central system administrator — The system administrator who is the `root` user (or the superuser) of the system. The `root` lnode is always the top of the scheduling tree. The central administrator has overall responsibility for the administration of all users and resources, but may delegate some administrative responsibility to other ordinary users by granting them administrative privileges. Typically, the central administrator determines the allocation of resources to the groups that are children of the `root` lnode, and grants administrative privileges to the group headers of each of those groups, thus relieving much of the associated administrative burden.

child — An lnode directly beneath another lnode in the scheduling tree. One lnode is a child of another if the first lnode's `sgroup` attribute is set to the UID of the second lnode. Correspondingly, the latter lnode is referred to as the parent or group header of the former.

decay — The periodic reduction of the usage of a renewable resource. For all resources except CPU usage, the decay is a fixed amount deemed to be subtracted from the usage attribute on a regular basis. For CPU usage, an exponential (multiplicative) decay is used.

effective share — An lnode's current effective share is determined by its allocated share, together with its `cpu.usage` attribute.

entitlement — The amount of CPU time allocated to a particular user.

field number The array slot(s) used by attributes, specified in the configuration file.

fixed resource A resource with a finite total supply.

flag A special type of attribute that is similar to a boolean variable, except that it may have one of four values: `set`, `clear`, `group`, or `inherit`. Flags are used within Solaris Resource Manager to control privileges.

group Within Solaris Resource Manager, this term normally refers to a scheduling group. See *scheduling group*.

group administrator The lnode at the head of a group, referred to as a group administrator or group header lnode. A group header has administrative privilege over the members of the scheduling group that he or she heads. Status is granted by setting the `flag.admin` flag of a group header. Group administrators are allowed to control resource and privilege allocation within their group, and to further delegate administrative responsibility to group headers within their group. They are not considered members of the group that they head.

group header See *group administrator*.

group loop When an lnode is made active, all of its parents up to the `root` lnode are also activated. As a result of this process, if one of the lnodes is seen to have a parent that has already been encountered, the kernel has discovered a group loop.

idle lnode A special lnode to which unused CPU time is accrued. This may be useful for accounting purposes. The default user name is `srmidle`, which has a UID of *41*.

inherit One of the possible values that may be given to a flag attribute. When a flag with an immediate value of `inherit` is being evaluated, the same flag in the parent lnode is evaluated to determine the actual value. This process is recursive. If the flag is set to `inherit` on the `root` lnode, then the final value is determined from the default value. The result of evaluating a flag is always `set` or `clear`.

kernel The core of the operating system; it supports system calls, file systems, and process scheduling. Solaris Resource Manager consists of two kernel modules and a number of kernel hooks, as well as user-level (non-kernel) programs and library routines.

leaf node An lnode with no children.

limit A numeric attribute associated with a usage attribute. A user's usage of a resource is prevented from exceeding the limits for that resource. There are two kinds of limits: hard and soft.

- A hard limit will cause resource consumption or allocation attempts to fail if they would cause the usage to exceed the limit.

- A soft limit typically does not directly constrain usage, but represents instead a point at which the user is informed of the usage and is encouraged to reduce it.

A limit of zero is a special case; it means that no limit applies.

limits database The database of user information that Solaris Resource Manager uses to perform all resource control. It contains one lnode per UID, which is accessed by using the UID as a direct index into the file.

lnode A limit node is a fixed-length structure used by Solaris Resource Manager to hold all per-user data required in addition to the data stored in the password map. It is a structure stored on disk in the limits database, and it is read and written by the kernel as required. There is at most a single lnode for each unique UID. Different accounts that have the same UID use the same lnode.

lnode database The on-disk copy of all the lnodes used by Solaris Resource Manager, indexed by UID.

lost lnode A special lnode used when the `setuid()` system call cannot attach a process to the lnode corresponding to the target UID of the system call, usually because that lnode does not exist. The default user name is `srmlost`, which has a UID of *42*.

notification Any message that is sent to the Solaris Resource Manager daemon, `limdaemon`. Some notification messages have special meaning to the `limdaemon`.

orphan An lnode whose parent lnode does not exist. That is, the UID specified in the `sgroup` attribute of the lnode does not itself correspond to an lnode.

other lnode If an account exists for a user named other, and an lnode exists for that account, then that lnode will be used as the default for the parent of lnodes newly created by the superuser or *uselimadm* users using the `limadm` command. The system-assigned name `srmother` cannot be changed.

page A portion of physical memory.

page in To read data from a file into memory one page at a time.

page out To relocate pages to an area outside of physical memory.

parent The group header of an lnode in the scheduling tree.

password map The Solaris enviroment database of user accounts maintained when the system is using a name service such as NIS or NIS+.

peer Another lnode within an lnode's scheduling group, excluding the parent of the group.

renewable resource A resource in which more units become available over time. For example, CPU usage or connect-time.

`root` lnode The lnode for UID 0. This lnode is the head of the entire tree of lnodes, making all other lnodes its members.

scheduling group Solaris Resource Manager allows all users to be organized into a system-wide hierarchy of scheduling groups that typically reflects the structure of the organizations using the system. The term "scheduling group" is used in preference to "group" to avoid confusion with the existing UNIX group concept, even though scheduling groups are used for much more than just scheduling. Solaris Resource Manager groups need bear no relation to the groups defined in the UNIX `/etc/group` file.

A scheduling group at any level in the lnode hierarchy can be treated as a single user. That is, resource limits assigned to a scheduling group apply to the net usage of all groups and users within that group.

scheduling tree The tree of lnodes, headed by the `root` lnode, with particular reference to the parent-child relationship between lnodes, the allocation of CPU shares, and the way in which the Solaris Resource Manager scheduler determines process run rates.

shares A way of defining the proportion of CPU entitlement that an lnode has with respect to its parent and peer lnodes. The concept of CPU shares is analogous to shares in a company; what matters is not how many you have, but how many compared with other shareholders.

SHR scheduler The module responsible for allocating resources according to the plan laid out in the administrative profile.

Solaris Resource Manager login session Any login-like connection to the system of which Solaris Resource Manager is aware. It requires cooperation between Solaris Resource Manager and the various 'gateway' programs that are responsible for authenticating users and granting them access.

sub-administrator Users who have the `flag.uselimadm` flag set. They are assistants to root, and they can execute any `limadm` command as root.

superuser A person who knows the `root` password. Processes have superuser privilege if they are running with an effective UID of 0.

topmost group Any group with `root` as its group header.

units The basic quantity of a given resource. Values within Solaris Resource Manager are represented as one of three types of units: scaled, raw, or internal.

usage A numeric attribute which increases whenever a user consumes or is allocated some of a resource. For fixed resources, the usage is

	decreased whenever any of the resource is freed. For renewable resources, the usage is decreased whenever a decay is performed.
`uselimadm` **user**	A user whose lnode has the flag `uselimadm` attribute set. This gives a user the same privileges (with regard to Solaris Resource Manager administration) as the superuser.
user attribute	See *attribute*.
user-mode	The way in which code is executed by normal programs and processes on a UNIX system. The alternative, kernel-mode, is used by system calls, device drivers, and the SYS class scheduler. Solaris Resource Manager has some components that run in user-mode and some that run in kernel-mode.