Solaris Containers — What They Are and How to Use Them

Menno Lageman, Sun Client Solutions

Sun BluePrints™ OnLine—May 2005

http://www.sun.com/blueprints

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054 U.S.A.
650 960-1300

Part No.819-2679-10
Revision 1.0, 4/28/05
Edition: May 2005

Please
Recycle

1

Executive Overview

Over the years businesses have been building large-scale information systems to solve business problems, with a focus on building scalable and highly available IT infrastructures that can adapt change. Providing sufficient availability and performance for business applications was the primary driver for these efforts. Today, the need to protect technology investments and provide the same service levels at a lower price point is shifting the focus to reducing IT infrastructure cost and improving end user service level management. Sun believes businesses can accomplish this goal by utilizing the facilities available in *Solaris Containers*.

End user application services are frequently comprised of components that are distributed across multiple servers. To reduce costs, businesses are eager to consolidate these applications onto fewer servers, but they must be careful to maintain isolation between applications. Hardware and software advances have led to the concept of *server virtualization*, the partitioning of network servers into several independent execution environments. Server virtualization allows a data center to be viewed and managed as a set of compute resources rather than a room of individual systems.

Solaris Containers

Sun's next advance in server virtualization is a concept called Solaris Containers. Solaris Containers provide isolation between software applications or services using flexible, software-defined boundaries. Applications can be managed independently of each other, even while running in the same instance of the Solaris™ Operating System (Solaris OS). Solaris Containers create an execution environment within a single instance of the Solaris OS and provide:

- *Full resource containment and control* for more predictable service levels
- *Software fault isolation* to minimize fault propagation and unplanned downtime
- *Security isolation* to prevent unauthorized access as well as unintentional intrusions

The primary benefits of Solaris Containers are:

- *Reduced management costs* through server consolidation, and a reduced number of operating system instances
- *Increased resource utilization* with dynamic resource reallocation between Containers
- *Increased service availability* by minimizing fault propagation and security violations between applications
- *Increased flexibility* because software based Containers can be dynamically reconfigured
- *Increased accuracy and flexibility of accounting*, based on workloads rather than systems or processes

How This Article is Organized

- Chapter 2, "Introduction," provides an overview of the features of Solaris Containers.
- Chapter 3, "Workload Management," describes the nature of workloads today and how resource management techniques can be used to gives those workloads the resources they need.
- Chapter 4, "Managing Workloads — An Example," provides an example of how to use projects to manage workloads effectively.
- Chapter 5, "Dynamic Resource Pools," describes how dynamic resource pools can be used to automatically allocate resources.
- Chapter 6, "Resource Pools — An Example," provides an example of how to use resource pools to partition the CPU resources on a system.
- Chapter 7, "Solaris Zones," describes the new Zones features in the Solaris 10 Operating System.
- Chapter 8, "Using Zones — An Example," illustrates how to use Zones to create virtual environments on a system.
- Chapter 9, "Containers — An Example," describes how to create Containers that can be used to consolidate multiple copies of applications onto a system.
- Chapter 10, "Summary," provides links to more information.

Typographic Conventions

TABLE 1-1 describes the typographic conventions used in this article.

TABLE 1-1 Typographic Conventions

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

2

Introduction

Today, businesses often design their systems with extra capacity to handle occasional peak loads to maximize revenue during periods of high demand. This extra system capacity remains unused during periods of normal demand. By allowing other applications to borrow this unused capacity a more cost-effective solution can be realized. During periods of high demand resources can be dynamically reallocated to important applications. Sharing resources in this way leads to higher resource utilization, reduces capital and system management costs by reducing the total number of systems required. For the consolidation of applications onto fewer systems to be effective, applications must be able to be managed independently. This requires the ability to control resource utilization, isolate faults, and manage security between multiple applications on the same server. In other words, it requires the establishment of virtual server boundaries within the server.

One of the first steps in this direction was the introduction of Dynamic System Domains on large Sun servers. With Dynamic System Domains, a server can be divided into several domains, each running its own copy of the Solaris OS. The domains provide hardware isolation between the applications so that faults in one domain do not propagate to applications in other domains. Domain boundaries can be dynamically partitioned to adapt to changing resource requirements. Resources can be moved from one domain to another without requiring a restart of the system. This adds flexibility to the data center while maintaining security and isolation from faults in other domains.

Starting with Solaris Resource Manager 1.x in the Solaris 2.6 OS, Sun has gradually enhanced the ability to control resource utilization and separate applications running in a single instance of the Solaris OS. Several technologies have been added to the Solaris OS over the years, providing additional capabilities and finer control over resource utilization. Examples of such technologies include the Solaris 9 Resource Manager and Resource Pools in the Solaris 9 OS. These technologies allow users to create a Solaris Container, an application or *service* that has one or more resource boundaries associated with it. These resource boundaries can limit CPU or memory consumption, network bandwidth, or even be a processor set. As a result, Solaris Containers are a prime enabler for server consolidation.

With the introduction of Solaris Zones in the Solaris 10 OS, Sun is taking Solaris Containers a step further by allowing servers to be partitioned in sub-CPU granularity. A Solaris Zone is a complete execution environment for a set of software services — a separate, virtual Solaris

environment within a Solaris instance. A Zone provides a virtual mapping from software services to platform resources, and allows application components to be isolated from each other even though they share a single Solaris OS instance. It establishes boundaries for resource consumption and provides isolation from other Zones on the same system. The boundaries can be changed dynamically to adapt to changing processing requirements of the applications running in the Zone.

Solaris Containers can be built using one or more the following technologies. These technologies can be combined to create Containers tailored for a specific server consolidation project.

- Solaris Resource Manager, for workload resource management
- Resource Pools, for partitioning
- Zones, for isolation, security and virtualization

It is important to note that a Solaris Container is not equivalent to a Solaris Zone. Zones technology can be used to create a Container with certain characteristics, such as the isolation provided by the virtual Solaris environment. But it is also possible to create another Solaris Container using Resource Pools technology if the required characterics of that Container can be met with the features Resource Pools provide. So while a Zone is a Container, a Container is not necessarily a Zone.

Workload Resource Management

One of the inhibitors for consolidating multiple applications onto a single server is the lack of control over the resources utilized by applications. Consider the example of a company that wants to consolidate two database servers onto one system to decrease the number of systems to manage, as well as the number of software licenses required.

The first database is used by an on-line sales application while the second database is used by a marketing application. Because the sales application supports the core business of the company, it should be guaranteed a certain minimum amount of CPU when needed. The marketing application is a supporting application, and the CPU requirements of the database server are much less stringent. Without a mechanism to enforce these requirements, these two

applications cannot be consolidated successfully onto one system. With Solaris Containers, these business requirements can be implemented by establishing the appropriate CPU resource boundaries using the *Fair Share Scheduler* (Figure 2-1).

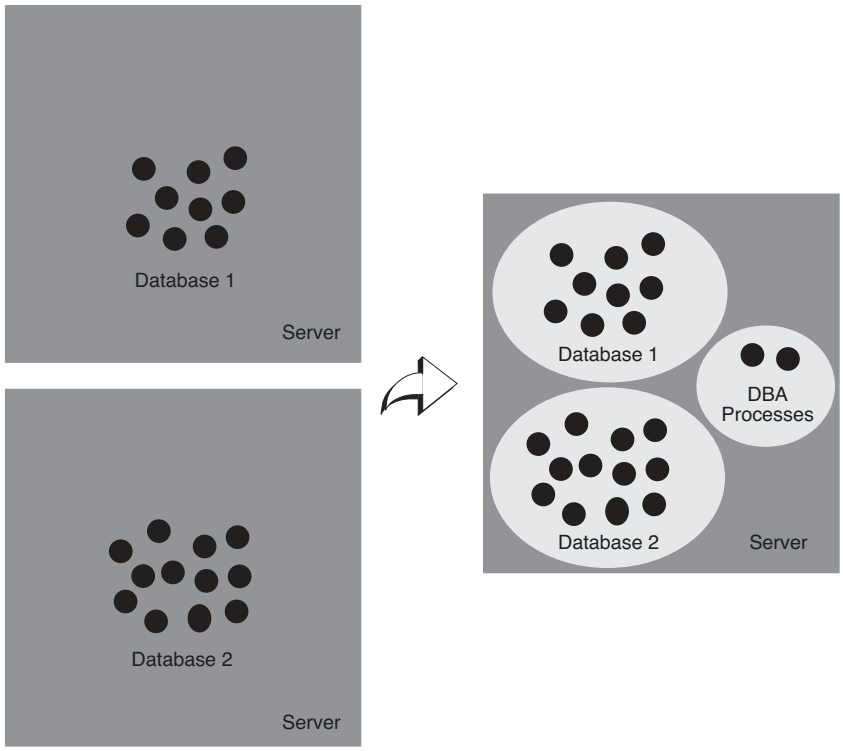

FIGURE 2-1 Solaris Containers provide an environment that fosters the safe consolidation of applications onto a single server

The Fair Share Scheduler controls the allocation of available CPU resources among workloads based on their relative importance. See Chapter 3 for more information on workload management and Chapter 4 for a hands-on example of workload management.

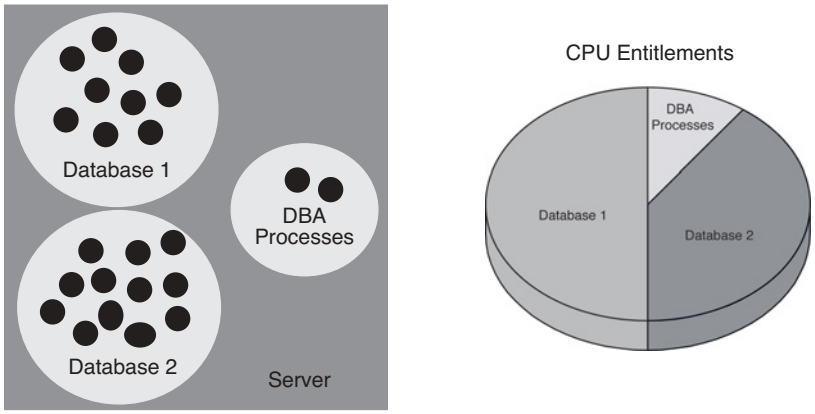

FIGURE 2-2 The Fair Share Scheduler ensures applications get the resources they need

Partitioning

In some cases a more strict separation of resource consumption may be required. For example, some applications may require a dedicated number of CPUs regardless of the processing requirements of other workloads. Furthermore, it may be desirable to restrict some applications to a maximum number of CPU resources as defined in a service level agreement. Dynamic Resource Pools can be used to provide this kind of partitioning.

In the example of the sales and the marketing database, two Resource Pools can be created: one with a large number of CPUs, and another with a small number of CPUs (Figure 2-3). The sales database would be assigned to the large pool and the marketing database to the small pool. The Solaris 10 OS adds the capability to dynamically adjust these resource allocations in response to application load changes in order to meet system performance goals set by administrators. See Chapter 5 for more information on dynamic resource pools, and Chapter 6 for an example of dynamic resource pools.

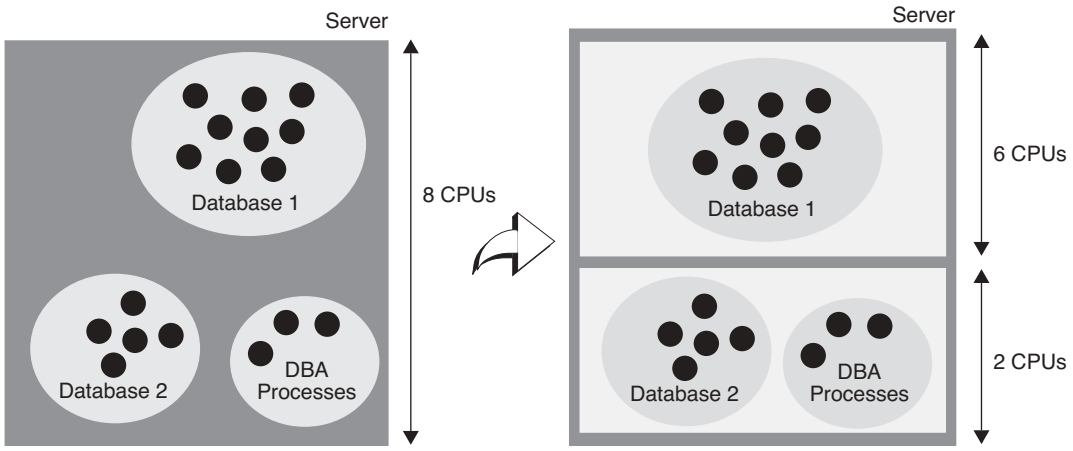

FIGURE 2-3 Resource pools can be used to partition resources

Isolation

Another inhibitor for consolidating applications is the lack of logical isolation between applications. This is of prime importance when the applications belong to different business units. Consider the example of an internal IT department acting as a service provider for a large corporation consolidating two workloads onto a single system. The IT department currently uses two systems, each dedicated to a single workload, since the workloads are from two different business units and each procured a system. Assuming the systems are currently under utilized, the organization wants to consolidate the applicatios onto a single system to achieve a more cost-effective solution.

However, both businesses object to sharing a system with another customer, as they are concerned about possible namespace conflicts, security issues, and administration conflicts. Solaris Containers make it possible to consolidate these workloads on one system by virtue of the namespace isolation, security and virtualization features of *Solaris™ Zones*. By creating a Zone for each business unit, the IT department can effectively create two separate systems on one physical system. To each business unit it appears as if they have a dedicated machine. See Chapter 7 for more information on Solaris Zones and Chapter 8 for a practical example of using Zones.

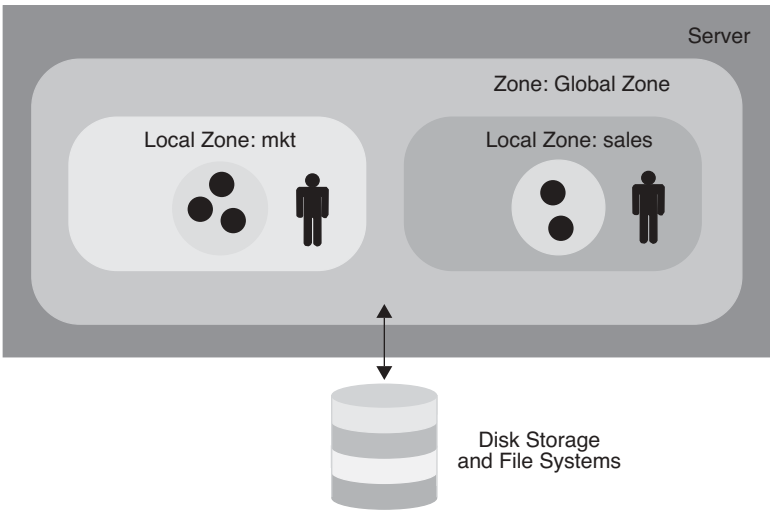

FIGURE 2-4 Zones effectively create separate systems on one physical system

Solaris Container Evolution

With the release of the Solaris 10 OS the Solaris Container reaches the full functionality discussed above. However, much of the functionality discussed in this document can be used in earlier Solaris OS releases. Some Solaris Containers features, such as the Fair Share Scheduler, have been available since the release of the Solaris 9 OS. Prior to that release, a fair share scheduler was available in the form of the Solaris Resource Manager 1.x unbundled software for Solaris 2.6 OS and later releases. This means that much of the functionality discussed in this document can be equally used on these earlier platforms. Appendix A contains a list of Container features and indicates the Solaris OS version in which they were first released.

Document Approach

To see how Solaris Containers can be used for consolidating applications this paper presents several technologies that help organizations implement Solaris Containers:

- Workload management
- Resource pools
- Solaris Zones

Each of these topics is discussed in two parts: a chapter that focuses on the concepts of the feature, followed by a chapter where the feature is used in a hands-on fashion using a simplified real world example. Readers that are interested solely in the concepts can skip the example chapters. By following the steps in the example chapters readers can recreate the examples on their own systems to experiment with a particular Solaris Containers feature. All examples can run on a uniprocessor system, with the exception of the processor sets portion of the resource pools example. This requires a multiprocessor system.

3

Workload Management

Running multiple applications on a single computer system without a means to control how applications use system resources can lead to unpredictable service levels. By default, the Solaris OS treats every resource request with equal priority. If there is enough of the resource available the request is granted. If the demand for the resource exceeds the total capacity available, the Solaris OS adapts by restricting access to the resource. The action taken to restrict access depends on the type of resource. For example, should demand for CPU time exceed the CPU time available, the scheduler reacts by adjusting the priorities of processes in order to change the distribution of the CPU time. The scheduler operates on threads and has no concept of applications, let alone their relative importance from a business perspective. An unimportant CPU-bound application can victimize other, more important applications by placing high demand for CPU resources on the system.

Other resources, such as the total number of processes on the system, have a fixed upper bound. Once the limit is reached, no more of this resource can be used. A runaway process that keeps creating new processes can prevent new useful work from being started. Other than specifying the system-wide upper limit, there is no way to limit the number of processes that may be created by an application or a set of applications.

What is needed is a way to control resource usage based on workloads. A *workload* is an aggregation of all processes of an application, or group of applications, that makes sense from a business perspective. Instead of managing resource usage at the process level, it should be possible to manage resource usage at the workload level. This allows the implementation of policies such as “the Sales application shall be granted at least 30% of CPU resources” as part of a service level agreement. The Solaris OS resource management features make it possible to treat workloads in this way by:

- Restricting access to specific resources
- Offering resources to workloads on a preferential basis
- Isolating workloads from each other

The first step in managing resource usage by workloads is identifying or classifying the components, such as processes, that make up the workload. The next step is measuring the resource consumption of these workloads. Finally, by applying constraints on the use of resources the workloads can be controlled. The constraints applied follow from the policies defined for the workloads based on business requirements.

A possible policy could be that an important workload should always be granted a minimum amount of CPU time even on an overloaded system. Another policy could be that a workload is only granted access to the CPU if there are no other workloads requiring CPU resources.

Projects

The first step in managing resource usage involves identifying the workloads running on the system. Possible approaches include identifying workloads by username or processname. While simple, this poses a challenge when multiple instances of the same application are running on the system for different workloads, such as a sales application database and a marketing application database. Unless the database application provides a way to run the instances as different users, it is imposible to attribute resource usage to a specific workload based solely on userid. In addition, aggregation of multiple related applications, such as database servers, application servers and Web servers for a business application on one system is not possible.

The Solaris OS provides a facility called *projects* to identify workloads. The project serves as an administrative tag used to group related work in a manner deemed useful by the system administrator. System administrators can, for example, create one project for the sales application and another project for the marketing application. By placing all processes related to the sales application in the sales project and the processes for the marketing application in the marketing project, the administrator can separate, and ultimately control, the workloads in a way that makes sense to the business.

A user that is a member of more than one project can run processes in multiple projects at the same time, making it possible for users to participate in several workloads simultaneously. All processes started by a process inherit the project of the parent process. As a result, switching to a new project in a startup script runs all child processes in the new project.

Using Projects to Define Workloads

In the example of the sales and marketing applications, the system administrator can create two new projects, one for the sales application and one for the marketing application. The application startup scripts must be modified to switch to the desired project as part of the application startup. The sales application startup script switches to the sales project, and the marketing application switches to the marketing project. This results in both applications

running in different projects while still using the same userid. Adding another application, such as a Web server, to the sales application workload requires adding the Web server user to the sales project and modifying the Web server startup script to switch to the sales project. With the introduction of the Service Management Facility (SMF) in the Solaris 10 OS, administrators can assign the project in which to run the application or service through service properties in the SMF repository.

The Project Database

Projects are defined in the project database. The project database can be a local file or in a name service such as NIS or LDAP. By putting the project database in NIS or LDAP, the project definition can be shared across multiple systems. Each entry in the project database consists of the following fields:

- **name**, the name of the project
- **id**, the project's unique numerical ID
- **comment**, the description of the project
- **user list**, a list of users allowed in the project
- **group list**, a list of groups allowed in the project
- **attributes**, a list of project attributes, such as resource controls

A freshly installed system always contains a local project database `/etc/project` containing five standard projects:

- `system`, used for all system processes and daemons
- `user.root`, used for all processes run by root
- `noproject`, for processes specific to IP quality of service (IPQoS)
- `default`, for users not matching any other project (a catch-all project)
- `group.staff`, for all users in the group `staff`

A user or group can be a member of one or more projects. The user and group lists in the project database determine in what projects a user or group of users can execute processes. These lists can contain wildcards to allow for flexible definitions, such as 'all members of group staff excluding user bob'. Users can switch to any project of which they are a member. Until the user changes the project in which to execute a process, all processes run in the user's default project. The user and group lists only define the project(s) in which a user or group is allowed to execute processes. It does not define a default project for the user or group. The default project for a user is determined by the system at login time. See the `man` page for `getprojent(3C)` for the exact algorithm used.

Commands

The following commands are available to administer projects:

Command	Description
`projadd(1M)`	Adds a new project to the local project database
`projmod(1M)`	Modifies a project entry in the local project database
`projdel(1M)`	Deletes a project entry from the local project database
`projects(1)`	Displays project membership for a user
`newtask(1)`	Switches to a project

Several standard Solaris OS commands include project related options, and can be used to view or manipulate processes based on their project membership:

Command	Option
`id(1M)`	`-p`
`ipcs(1)`	`-J`
`pgrep(1)`	`-J -T`
`pkill(1)`	`-J -T`
`poolbind(1M)`	`-i` *project*
`prctl(1)`	`-i` *project*
`priocntl(1M)`	`-i` *project*
`prstat(1M)`	`-j -J -k -T`
`ps(1)`	`-o` *projid project taskid*
`useradd(1M)`	`-p`

For example, the `prstat -J` command lists all processes and projects on the system and displays a per project total. See the `man` pages for more information on these commands and the options related to projects.

Extended Accounting

Once workloads are identified and labeled using projects, the next step in managing resource usage involves measuring workload resource consumption. While current consumption can be measured using the `prstat(1M)` command to obtain real-time snapshot of resource usage, it does not provide the capability to look at historical data.

The traditional accounting mechanism is process based and predates the introduction of projects. It is therefore unable to provide resource usage statistics based on workloads. The extended accounting facility allows collection of statistics at the process level, the task level or both. Accounting at the task level aggregates the resource usage of its member processes, thereby reducing the required disk space for accounting data. A *task* is a group of related processes executing in the same project as a result of a `newtask(1)` command. An accounting record is written at the completion of a process or task. Interim accounting records can be written for tasks, and can be used to provide accurate daily accounting for long running jobs that span multiple days.

Every process that runs in the system is associated with a project and a task. By labeling all resource usage records with the project for which the work was done, the extended accounting facility can provide data on the resource consumption of workloads. This data can be used for reporting, capacity planning or charge back schemes.

Unlike the traditional System V accounting mechanism that is based on fixed size, fixed semantic records, the extended accounting facility uses a flexible and extensible file format for accounting data. Files in this format can be read or written using the C language API provided by `libexacct(3LIB)`. This API abstracts the accounting file and offers functions to read and write records and fields in the file without the need for knowledge of the physical layout. This makes it possible to add new record or field types to the file between releases, even during system operation, without impacting existing applications that use extended accounting files. A Perl interface for `libexacct` is available to ease the creation of custom reporting tools.

Commands

The following commands are available to administer the extended accounting facility.

Command	Description
`acctadm(1M)`	Configure extended accounting
`wracct(1M)`	Write extended accounting records for active processes and tasks

The Fair Share Scheduler

Running multiple workloads on the same system can lead to a situation where one workload monopolizes CPU resources and impacts other workloads. This may result in important workloads not receiving sufficient CPU resources to complete their work. It is desirable to have a mechanism by which system administrators can prioritize access to CPU resources based on the importance of the workload.

The policy of the default scheduler in the Solaris OS is to give every process relatively equal access to CPU resources. Since it has no knowledge of workloads, the default scheduler cannot prioritize CPU allocation based on workload importance. The Solaris OS offers an

alternative scheduler that is aware of workloads and can prioritize CPU allocation with respect to workload importance.

CPU Shares

The *Fair Share Scheduler* (FSS) controls allocation of CPU resources using *CPU shares*. The importance of a workload is expressed by the number of shares the system administrator allocates to the project representing the workload. The Fair Share Scheduler ensures that CPU resources are distributed among active projects based on the number of shares assigned to each project (Figure 3-1).

A CPU share defines a relative entitlement of the CPU resources available to a project on the system. It is important to note that CPU shares are *not* the same as CPU percentages. Shares define the *relative importance* of projects with respect to other projects. If project A is deemed twice as important as project B, project A should be assigned twice as many shares as project B. The actual number of shares assigned is largely irrelevant — two shares for project A versus one share for project B yields the same results as 18 shares for project A versus nine shares for project B. In both cases, Project A is entitled to twice the amount of CPU resources as project B. The importance of project A relative to project B can be increased by assigning more shares to project A while retaining the same number of shares for project B.

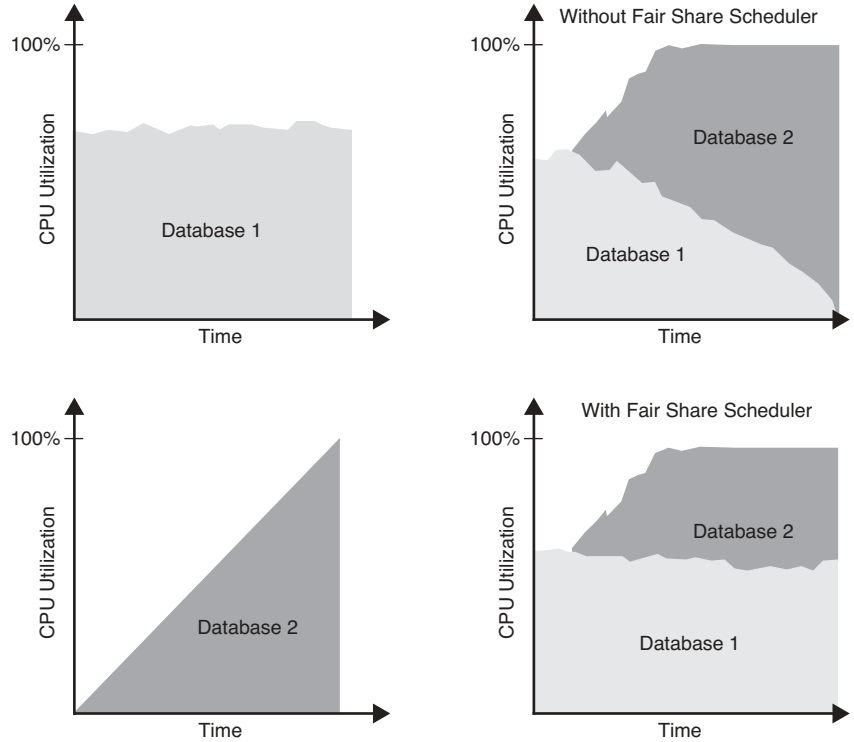

FIGURE 3-1 The Fair Share Scheduler ensures applications get the CPU resources to which they are entitled.

The Fair Share Scheduler calculates the proportion of CPU resources allocated to a project by dividing the shares for the project by the total number of shares of active projects. An *active project* is a project with at least one process using CPU resources. Shares for idle projects, such as those without active processes, are not used in the calculations. For example, consider projects A, B and C with two, one and four shares respectively. If projects A, B and C are active, then project A is entitled to $\frac{2}{7}$, project B is entitled to $\frac{1}{7}$, and project C is entitled to $\frac{4}{7}$ of CPU resources. If project A is idle, project B is entitled to $\frac{1}{5}$ of CPU resources, and project C is entitled to $\frac{4}{5}$ of CPU resources (Figure 3-2). Note that even though the actual CPU entitlement for project B and C increases, the proportion between project B and C stays the same (1:4).

It is important to note that the Fair Share Scheduler only limits CPU usage if there is competition for CPU resources. If there is only one active project on the system, it can use 100% of CPU resources, regardless of the number of shares it holds. CPU cycles are never wasted. If a project does not use all the CPU resources it is entitled to because it has no work to do, the remaining CPU resources are distributed between other active projects.

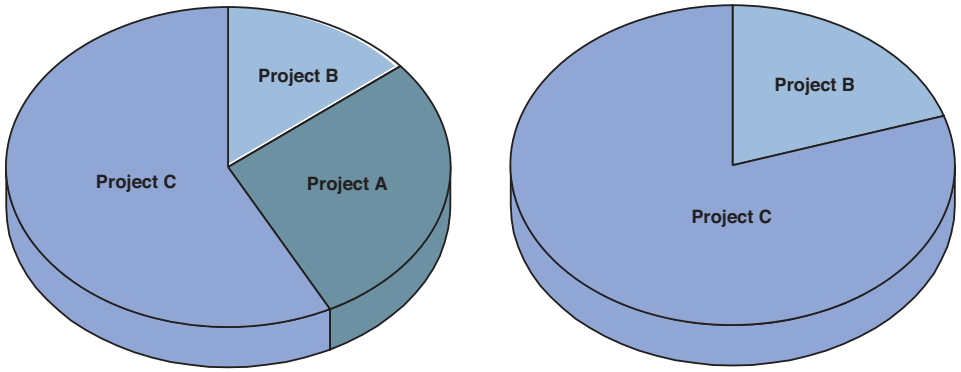

FIGURE 3-2 The Fair Share Scheduler distributes CPU resources among active projects based on the number of CPU shares

CPU Shares Configuration

CPU shares are configured through the `project.cpu-shares` resource control in the project database. Every project can be assigned a `project.cpu-shares` resource control. Projects without this resource control are assigned one share by the system. The `system` project is used for all system processes and daemons, and is special in that it has unlimited shares. Projects with zero shares assigned are only allowed to run when no other projects with non-zero shares are active.

Users can be a member of multiple projects and CPU usage is controlled by the number of shares of the project in which the user executes. As a result, a user can be entitled to different amounts of CPU resources at the same time. Note that a process can only be in one project at a time, so having different amounts of CPU resources at the same time means that processes owned by this user reside in different projects.

To place a CPU usage limit on a single user, create a project with the appropriate number of shares that contains only that user. This project should be the default project for this user, and the user should not be a member of any other projects to prevent the user from switching to another project.

The CPU shares can be adjusted dynamically using the `prctl(1M)` command. These changes are valid until the next system boot. To make the changes permanent, update the `project.cpu-shares` resource control in the project database.

Resource Controls

Resource usage of workloads can be controlled by placing bounds on resource usage. These bounds can be used to prevent a workload from over-consuming a particular resource and interfering with other workloads. The Solaris OS provides a resource controls facility to implement constraints on resource consumption. This facility is an extension of the traditional UNIX resource limit facility (*rlimit*). The rlimit facility can be used to set limits on the resource usage of processes, such as the maximum CPU time used, the maximum file size, the maximum core file size, and more. However, as the rlimit facility is process-based, its use for constraining workloads is rather limited. The resource controls facility in the Solaris OS extends process-based limits by adding resource limits at the task and project level. The number of resource limits that can be set is also expanded to give system administrators more control over resource consumption by processes, tasks and projects on the system.

Administering Resource Controls

Resource controls are configured through the project database. The last field of the project entry is used to set resource controls. A resource control in the project entry is a name-value pair. The name denotes the type of limit, while the value is a list of attributes for the control. Multiple resource controls can be added to a single project entry by separating the resource controls with a semicolon. The list of attributes for a resource control consists of a privilege level, a threshold, and an action.

The privilege level determines which users can modify the threshold value. Three privilege levels are provided:

- **basic**, the owner of the calling process can change the threshold
- **privileged**, only privileged (superuser) users can change the threshold
- **system**, the threshold is fixed for the lifetime of the operating system instance

Every resource control has at least a system value, which represents how much of the resource the current implementation of the operating system is able to provide. A resource control can have at most one basic value and any number of privileged values.

The action defines the steps to be taken when the threshold is exceeded. Three actions are possible:

- **deny**, deny resource requests for an amount that is greater than the threshold
- **signal**, send the specified signal to the process exceeding the threshold value
- **none**, perform no action when the threshold is exceeded

Note – Changes made in the project database are only applied when a new process, task or project starts. Existing processes, tasks and projects do not see these changes. The `prctl(1M)` and `rctladm(1M)`commands can be used to change resource controls on active entities.

Available Resource Controls

The following table identifies the resource controls available in the Solaris 10 OS.

Resource Control	Description
process.max-port-events	Maximum allowable number of events per event port
process.max-msg-messages	Maximum number of messages on a message queue
process.max-msg-qbytes	Maximum number of bytes of messages on a message queue
process.max-sem-ops	Maximum number of semaphore operations allowed per semop call
process.max-sem-nsems	Maximum number of semaphores allowed per semaphore set
process.max-address-space	Maximum amount of address space available to this process
process.max-file-descriptor	Maximum file descriptor index available to this process
process.max-core-size	Maximum size of a core file created by this process
process.max-stack-size	Maximum stack memory segment available to this process
process.max-data-size	Maximum heap memory available to this process
process.max-file-size	Maximum file offset available for writing by this process
process.max-cpu-time	Maximum CPU time available to this process
task.max-cpu-time	Maximum CPU time available to this task's processes
task.max-lwps	Maximum number of LWPs simultaneously available to tasks's processes
project.max-contracts	Maximum number of contracts allowed in a project
project.max-device-locked-memory	Total amount of locked memory allowed in a project
project.max-port-ids	Maximum allowable number of event ports
project.max-shm-memory	Total amount of shared memory allowed for a project
project.max-shm-ids	Maximum number of shared memory IDs allowed for a project
project.max-msg-ids	Maximum number of message queue IDs allowed for a project
project.max-sem-ids	Maximum number of semaphore IDs allowed for a project
project.max-crypto-memory	Total amount of kernel memory that can be used by libpkcs11 for hardware crypto acceleration
project.max-tasks	Maximum number of tasks allowable in a project
project.max-lwps	Maximum number of LWPs simultaneously available to a project
project.cpu-shares	Number of CPU shares granted to a project for use with the FSS
zone.max-lwps	Maximum number of LWPs simultaneously available to zone's processes
zone.cpu-shares	Number of CPU shares granted to a zone for use with the FSS

Determining Thresholds

The resource consumption of processes is often unknown, so choosing a useful and safe threshold for a resource control can be a difficult task. Selecting an arbitrary threshold can lead to unexpected application failure modes. While some required information could be extracted from extended accounting information, there is a simpler way. The resource controls facility provides a global log action that sends a message to syslog when a threshold is exceeded.

First, a resource control with the threshold value to be verified must be set. The action should be set to ‘none’ to ensure the resource is not denied if the threshold is exceeded. This allows the process to run unconstrained. Next, the global syslog action for the resource control must be enabled. When the application exceeds the threshold for that resource control, a message that the resource control threshold has been exceeded is logged to syslog. By changing the threshold until the warning no longer appears during *normal* use of the application, a reasonable setting for the resource control can be determined. After determining the value for the resource control, the action should be changed to ‘deny’, to ensure the threshold is enforced by the system.

Commands

The following commands are available for administering resource controls. More information can be found in the `man` pages for each command.

Command	Description
`prctl(1M)`	Get or set resource controls on a running process, task or project
`rctladm(1M)`	Display or modify global state of system resource controls

4

Managing Workloads — An Example

Introduction

To demonstrate the concepts explained in the previous chapter, this chapter uses the Solaris OS resource management facilities to manage workloads on an example system. The system is shared by several business units and is running two workloads: two database instances, one for a marketing application and one for a sales application.

A project is defined for each workload, enabling the Fair Share Scheduler to be used to manage CPU allocation between the workloads. A resource control is added to limit the amount of shared memory for each workload. To account for all activity of the `oracle` user that is not related to either of these workloads, a third project is created. This project is the default project for the `oracle` user.

Requirements

The following minimum requirements are needed to run this example:

- Oracle 9i media (version 9.2.0.1.0)
- 6 GB disk space for the Oracle binaries and databases

Defining the Projects

To keep things simple, a local `/etc/project` database is used. The project entry in the `/etc/nsswitch.conf` file should be defined as follows:

```
# cat /etc/nsswitch
...
project:    files
...
```

By convention, Oracle instances are run as the user `oracle` in group `dba`. As a result, the group `dba` and user `oracle` are created:

```
# groupadd dba
# mkdir -p /export/home
# useradd -g dba -d /export/home/oracle -m -s /bin/bash oracle
```

A project named `group.dba` is created to serve as the default project for the user `oracle`. The system uses the rules described in the `getprojent(3C)` man page to determine the default project when a user logs in. Since the default group of user `oracle` is the dba group, the `group.<groupname>` rule matches and the `group.dba` project is set as the default project for user `oracle`. A comment describing the project is added using the `-c` option:

```
# projadd -c "Oracle default project" group.dba
```

The `id(1M)` command can be used to verify the default project for the `oracle` user:

```
# su - oracle
$ id -p
uid=100(oracle) gid=100(dba) projid=100(group.dba)
$ exit
```

To manage each Oracle instance as a separate workload, a project is created for each Oracle instance to run in: project `ora_mkt` for the marketing Oracle instance, and project `ora_sales` for the sales Oracle instance.

```
# projadd -c “Oracle Marketing” -U oracle ora_mkt
# projadd -c “Oracle Sales” -U oracle ora_sales
```

The `-U oracle` option specifies that the `oracle` user is allowed to run processes in these projects. Once these steps are complete, the `/etc/project` file contains the following information:

```
# cat /etc/project
system:0::::
user.root:1::::
noproject:2::::
default:3::::
group.staff:10::::
group.dba:100:Oracle default project:::
ora_mkt:101:Oracle Marketing:oracle::
ora_sales:102:oracle Sales:oracle::
```

The first five projects are projects that are created during system installation. Note that the system assigned project IDs for the last three projects since they were not explicitly specified on the `projadd` command.

System V IPC Resource Controls

The System V IPC resource limits in the Solaris 10 OS, such as the maximum shared memory size, are no longer set in the `/etc/system` file, but instead are project resource controls. As a result, a system reboot is not longer required to put changes to these parameters in effect. This also allows system administrators to set different values for different projects. A number of System V IPC parameters are obsolete with the Solaris 10 OS, simply because they are no longer necessary. The remaining parameters have more reasonable defaults to enable more applications to work out-of-the-box, without requiring these parameters to be set. The following table identifies the values recommended by the Oracle Installation Guide and the corresponding Solaris OS resource controls.

Parameter	Oracle Recommendation	Resource Control	Default Value
SEMMNI (semsys:seminfo_semmni)	100	project.max-sem-ids	128
SEMMNS (semsys:seminfo_semmns)	1024	obsolete	
SEMMSL (semsys:seminfo_semmsl)	256	project.max-sem-nsems	512
SHMMAX (shmsys:shminfo_shmmax)		project.max-shm-memory	1/4 physical memory
SHMMIN (shmsys:shminfo_shmmin)	1	obsolete	
SHMMNI (shmsys:shminfo_shmmni)	100	project.max-shm-ids	128
SHMSEG (shmsys:shminfo_shmseg)	10	obsolete	

Since the default values are higher than Oracle recommended values, the only resource control that must be set is `project.max-shm-memory`. To set the maximum shared memory size to 2 GB, add the `project.max-shm-memory=(privileged,2147483648,deny)` resource control to the last field of the project entries for the three Oracle projects.

```
# projmod -sK "project.max-shm-memory=(privileged,2G,deny)" group.dba
# projmod -sK "project.max-shm-memory=(privileged,2G,deny)" ora_mkt
# projmod -sK "project.max-shm-memory=(privileged,2G,deny)" ora_sales
```

Once these steps are complete, the `/etc/project` file should contain the following. Note that changes are shown in italics.

```
# cat /etc/project
system:0::::
user.root:1::::
noproject:2::::
default:3::::
group.staff:10::::
group.dba:100:Oracle default project:::project.max-shm-
memory=(privileged,2147483648,deny)
ora_mkt:101:Oracle Marketing:oracle::project.max-shm-memory=(privileged,2147483648,deny)
ora_sales:102:oracle Sales:oracle::project.max-shm-memory=(privileged,2147483648,deny)
```

To verify that the resource control is active, the `id(1M)` and `prctl(1)` commands can be used.

```
# su - oracle
$ id -p
uid=100(oracle) gid=100(dba) projid=100(group.dba)
$ prctl -n project.max-shm-memory -i process $$
process: 5754: -bash
NAME    PRIVILEGE       VALUE    FLAG   ACTION
RECIPIENT
project.max-shm-memory
        privileged      2.00GB      -   deny
```

Logging in as the `oracle` user creates a new task in the `group.dba` project, causing the entry in the project database to be read and the resource control to be set. As can be seen in the fifth line of output from the `prtcl` command, a resource control limiting the maximum shared memory size for the project to 2 GB is present.

Installing Oracle and Creating the Databases

Oracle installation consists of a series of steps, including software installation and the creation of `smf(5)` services for the Oracle instances. The procedure for installing Oracle is described in Appendix B on page 91.

In this example, a directory `/u01` with at least 6 GB of free space is required for the Oracle software and databases. A simple database is created for each workload. These databases are created using the procedure described in Appendix C on page 93. Use the database identifiers listed in the table below.

Database	Database Identifier (ORACLE_SID)
Marketing	MKT
Sales	SALES

Running Oracle Instances in Different Projects

The Oracle instances must run in separate projects in order to control them as separate entities using the Solaris Resource Manager. The processes of the marketing database instance should run in project `ora_mkt`, and the processes of the sales database instance should run in the `ora_sales` project. Since the Oracle provided start scripts are not project-aware, the processes of both instances run in the default project of the Oracle user `group.dba`. To run the instances in different projects, the Oracle start scripts must be made project-aware by issuing `/usr/bin/newtask -p ora_sales` as part of the startup of the sales database instance. This moves the current process and its childeren to the `ora_sales` project.

The Service Management Facility (SMF) in the Solaris 10 OS replaces the traditional way of managing application startup and shutdown through run control scripts. SMF uses a concept called *services* to accomplish this task. An SMF service consists of a set of methods and properties that describe service behavior. Examples of methods include the start and stop methods that `smf(5)` calls to start or stop the service. Properties are used to describe the service, such as dependencies on other required services, the user to run the service as, and the project in which to run the service. Through a set of `smf(5)` commands, services can be managed in a consistent manner. See the *System Administration Guide: Basic Administration* for more information on the Service Management Facility.

To run the example Oracle database instances in separate projects, two simple SMF services must be created: a `salesdb` service and `mktdb` service.

The service for the sales database is created by importing the manifest for the service into the SMF repository. By convention, manifests for site-specific services are placed in the directory `/var/svc/manifest/site`. A manifest is an XML file that defines service properties and methods. One of the properties of an SMF service is the user under which the service should run. In this example, the user is `oracle`. The project in which the service should run is also a service property. In this example, the project is `ora_sales`. The relevant part of the manifest is shown below. The full manifest for the sales database and marketing database services can be found in Appendix D on page 95.

```
# cd /var/svc/manifest/site
# cat salesdb.xml
[...]
        <exec_method
                type='method'
                name='start'
                exec='/u01/app/method/ora start SALES'
                timeout_seconds='0'>
                <method_context
                        project='ora_sales'>
                        <method_credential user='oracle' />
                </method_context>
        </exec_method>
[...]
```

The project attribute of the `method_context` element determines the project in which the service runs. The user attibute of the `method_credential` element determines the user under which the service runs. The manifest for the marketing database service is equivalent except that its project attibute is set to `ora_mkt`.

The start and stop methods for both services are implemented in a single shell script (`/u01/app/method/ora`). The start method calls the script with `start` as the first argument, while the stop method calls the script with `stop` as the first argument. The Oracle database identifier is passed as the second argument.

```
# cat /u01/app/method/ora
#!/bin/sh
#
# Usage: ora ‘start’ | ‘stop’ db_id
#
ORACLE_SID=$2
ORACLE_HOME=/u01/app/oracle/product/9.2.0.1.0
export ORACLE_SID ORACLE_HOME

case “$1” in
                'start')
                $ORACLE_HOME/bin/sqlplus "/ as sysdba" <<START_EOF
startup
START_EOF
                ;;
                'stop')
                $ORACLE_HOME/bin/sqlplus "/ as sysdba" <<STOP_EOF
shutdown immediate
STOP_EOF
                ;;
esac
exit 0
```

The services are created by importing the manifest and subsequently enabling the services. Note that enabling a service implies a start of the service. The `ps(1)` command can be used to verify the intances are running in different projects.

```
# svccfg import salesdb.xml
# svccfg import mktdb.xml
# svcadm enable salesdb
# svcadm enable mktdb
# ps -u oracle -o user,project,comm
    USER  PROJECT COMMAND
  oracle ora_sales ora_lgwr_SALES
  oracle ora_sales ora_smon_SALES
  oracle ora_mkt   ora_smon_MKT
  oracle ora_sales ora_pmon_SALES
  oracle ora_sales ora_dbw0_SALES
  oracle ora_mkt   ora_ckpt_MKT
  oracle ora_sales ora_ckpt_SALES
  oracle ora_mkt   ora_lgwr_MKT
  oracle ora_mkt   ora_pmon_MKT
  oracle ora_mkt   ora_dbw0_MKT
  oracle ora_sales ora_reco_SALES
  oracle ora_mkt   ora_reco_MKT
```

The processes for the marketing database instance run in the `ora_mkt` project, the processes for the Sales database instance run in the `ora_sales` project.

Controlling CPU Consumption

Now that the Oracle instances are running in different projects, the Fair Share Scheduler can be used to control CPU consumption by the instances. Because the Fair Share Scheduler is not the default scheduler, it must be enabled using the `dispadmin(1M)` command:

```
# dispadmin -d FSS
```

The `dispadmin` command configures the Fair Share Scheduler (FSS) as the default scheduler to be enabled on the next reboot. It is possible to change to the Fair Share Scheduler without a reboot by moving all processes in the TS scheduler class and the `init(1M)` process to the

FSS scheduler class using the `prioctl(1M)` command. This change persists only until the next reboot, and the `dispadmin -d FSS` command is required to make the change permanent.

```
# priocntl -s -c FSS -i class TS
# priocntl -s -c FSS -i pid 1
```

The change of the scheduler class can be verified using the `ps(1)` command with the `-cafe` options. In the output below, the fourth column (marked CLS) shows that the Fair Share Scheduler (FSS) is now the scheduler for the processes:

```
# ps -cafe
     UID   PID  PPID  CLS PRI    STIME TTY         TIME CMD
    root     0     0  SYS  96   Dec 01 ?           0:01 sched
    root     1     0  FSS  29   Dec 01 ?           0:00 /etc/init -
    root     2     0  SYS  98   Dec 01 ?           0:00 pageout
    root     3     0  SYS  60   Dec 01 ?           9:45 fsflush
    root   556     1  FSS  29   Dec 01 ?           0:00 /usr/lib/saf/sac -t 300
...
  oracle  1967     1  FSS  29 11:03:35 ?           0:00 ora_dbw0_MKT
  oracle  1971     1  FSS  29 11:03:36 ?           0:00 ora_ckpt_MKT
  oracle  2002     1  FSS  29 11:03:47 ?           0:01 ora_smon_SALES
  oracle  1973     1  FSS  29 11:03:36 ?           0:01 ora_smon_MKT
  oracle  1965     1  FSS  29 11:03:35 ?           0:00 ora_pmon_MKT
  oracle  1996     1  FSS  29 11:03:46 ?           0:00 ora_dbw0_SALES
  oracle  1975     1  FSS  29 11:03:36 ?           0:00 ora_reco_MKT
  oracle  1998     1  FSS  29 11:03:47 ?           0:00 ora_lgwr_SALES
  oracle  1969     1  FSS  29 11:03:36 ?           0:00 ora_lgwr_MKT
  oracle  2000     1  FSS  29 11:03:47 ?           0:00 ora_ckpt_SALES
  oracle  1994     1  FSS  29 11:03:46 ?           0:00 ora_pmon_SALES
  oracle  2004     1  FSS  29 11:03:47 ?           0:00 ora_reco_SALES
....
```

The final step involves assigning CPU shares to the projects to control CPU consumption. Assuming that the sales database is twice as important as the marketing database, and should therefore be entitled to twice the amount of CPU resources, the number of CPU shares for the `ora_sales` project is set to twice the number of shares for the `ora_mkt` project. The other projects are assumed to be less important, and their shares remain at system assigned default values. To give the `ora_sales` and `ora_mkt` projects a higher proportion of CPU resources with respect to these projects, the CPU shares are chosen to be much larger than those for the other projects. These values entitle the `ora_sales` project to twenty times more CPU resources than the `group.dba` project, and twice as many as the `ora_mkt` project.

Project	CPU Shares
ora_sales	20
ora_mkt	10
group.dba	1 (default)
system	Unlimited
user.root	1 (default)
default	1 (default)
group.staff	1 (default)

The CPU shares are set using the `prctl(1M)` command:

```
# prctl -n project.cpu-shares -r -v 10 -i project ora_mkt
# prctl -n project.cpu-shares -r -v 20 -i project ora_sales
```

The current value of the `project.cpu-shares` resource control for a project can be checked as follows:

```
# prctl -n project.cpu-shares -i project ora_mkt
project: 101: ora_mkt
NAME    PRIVILEGE       VALUE    FLAG   ACTION                       RECIPIENT
project.cpu-shares
        privileged         10       -   none                             -
        system          65.5K     max   none                             -
# prctl -n project.cpu-shares -i project ora_sales
project: 102: ora_sales
NAME    PRIVILEGE       VALUE    FLAG   ACTION                       RECIPIENT
project.cpu-shares
        privileged         20       -   none                             -
        system          65.5K     max   none                             -
```

To make these values persistent, the `project.cpu-shares` resource controls must be added to the project database.

```
# projmod -sK “project.cpu-shares=(privileged,10,none)” ora_mkt
# projmod -sK “project.cpu-shares=(privileged,20,none)” ora_sales
```

```
# cat /etc/project
system:0::::
user.root:1::::
noproject:2::::
default:3::::
group.staff:10::::
group.dba:100:Oracle DBA:::project.max-shm-memory=(privileged,2147483648,deny)
ora_mkt:101:Oracle Marketing:oracle::project.cpu-
shares=(privileged,10,none);project.max-shm-memory=(privileged,2147483648,deny)
ora_sales:102:Oracle Sales:oracle::project.cpu-
shares=(privileged,20,none);project.max-shm-memory=(privileged,2147483648,deny)
```

Note – A project entry must be on one line. The above lines have been wrapped for readability. They should be on one line.

For demonstration purposes, the `nspin` utility is used to create enough CPU demand to show the Fair Share Scheduler in action[1]. The `nspin` utility is part of the Solaris Resource Manager 1.x software, and is available for download at `http://www.sun.com/bigadmin/software/nspin/nspin.tar.gz`. To create more demand for CPU resources than are available on the 4 CPU machine used here, four copies of `nspin` are run in both the `ora_mkt` and `ora_sales` projects.

```
$ id -p
uid=100(oracle) gid=100(dba) projid=100(group.dba)
$ newtask -p ora_mkt
$ nspin -n 4 &
[1] 2059
$ newtask -p ora_sales
$ id -p
uid=100(oracle) gid=100(dba) projid=102(ora_sales)
$ nspin -n 4 &
[1] 2066
```

The `newtask(1)` command is used to switch from the default `group.dba` project to the `ora_mkt` and ora_sales projects to run `nspin`. The `prstat(1M)` command can be used to show CPU utilization per project and verify that the Fair Share Scheduler is distributing CPU resources to the projects according to their CPU shares.

1. Any application that consumes large quantities of CPU resources can be used.

```
$ prstat -J
   PID USERNAME  SIZE   RSS STATE  PRI NICE      TIME  CPU PROCESS/NLWP
  2069 oracle   1064K  592K run      1    0   0:01:57  25% nspin/1
  2066 oracle   1072K  664K run     18    0   0:01:31  17% nspin/1
  2067 oracle   1072K  600K cpu1    30    0   0:01:05  12% nspin/1
  2068 oracle   1072K  600K run     28    0   0:01:06  12% nspin/1
  2061 oracle   1072K  600K run     17    0   0:01:31 8.7% nspin/1
  2059 oracle   1072K  664K run     17    0   0:01:07 8.3% nspin/1
  2060 oracle   1072K  600K cpu0    24    0   0:01:06 8.2% nspin/1
  2062 oracle   1064K  592K cpu3    18    0   0:01:13 7.9% nspin/1
  2058 root     6056K 5040K cpu2    59    0   0:00:00 0.0% prstat/1

PROJID    NPROC  SIZE   RSS MEMORY      TIME  CPU PROJECT
   102       11 1011M  712M    36%   0:05:40  66% ora_sales
   101       11 1011M  703M    36%   0:04:58  33% ora_mkt
     1        5   14M 9064K   0.4%   0:00:01 0.0% user.root
   100        1 2760K 1952K   0.1%   0:00:00 0.0% group.dba
     0       28   84M   23M   1.1%   0:00:23 0.0% system

Total: 56 processes, 196 lwps, load averages: 7.30, 3.09, 1.21
```

The top portion of the `prstat` display shows active processes sorted by CPU utilization. The bottom portion shows the statistics aggregated by project. The `ora_sales` project is receiving 66% of CPU resources, and the `ora_mkt` project is receiving 33%, even though both projects requested the same amount of CPU (four runnable `nspin` processes in each project). The Fair Share Scheduler allocates CPU resources according to the proportion of CPU shares of the active projects (using CPU time). The only active projects at the time are `ora_mkt` and `ora_sales`. As a result, the CPU entitlement for the `ora_sales` project equals (20/(20 + 10)) * 100 = 67%, while `ora_mkt` is entitled to (10/(20 + 10)) * 100 = 33%. This matches the actual CPU usage observed using `prstat(1M)`.

Using Extended Accounting

Resource usage per project can be obtained using the Extended Accounting facility of the Solaris OS. Accounting records can be written per process, per task or both. To obtain resource usage per project, task accounting is sufficient. Rather than summarizing all process termination records from the process accounting file, task accounting files can be used instead. This involves substantially fewer records since the task accounting files consolidate multiple process records into one task record. Because tasks usually have a long life span and task accounting records are only written at the end of a task, *interval records* can be used to obtain accurate daily accounting. An interval record writes the current task usage to the

accounting file and resets the task usage to zero. The total task usage is the sum of all interval records plus the termination record. Examples of common long running tasks include HPC jobs and database processes.

Extended Accounting is turned off by default, and must be turned on using the `acctadm(1M)` command. In this example, the accounting files are named `task`*yyymmdd*. A `cron(1M)` job is used to switch files every night at midnight. To start the extended accounting facility at system boot time, a link to `/etc/init.d/acctadm` must be created in `/etc/rc2.d`:

```
# acctadm -f /var/adm/exacct/task`date '+%y%m%d'` task
# ln -s /etc/init.d/acctadm /etc/rc2.d/S01acctadm
```

The following script writes interval records for all tasks and then switches to a new accounting file:

```
# cat /opt/local/bin/switchexacct
#!/bin/sh
#
# Write interval record for all active tasks and switch accounting file
#
PATH=/usr/bin:/usr/sbin
wracct -i "`ps -efo taskid= | sort -u`" -t interval task
acctadm -f /var/adm/exacct/task`date '+%y%m%d'` task
```

Add the following line to the crontab of the root user to execute the `switchexacct` script at 00:00:

```
0 0 * * * /opt/local/bin/switchexacct > /dev/null 2>&1
```

The following script uses the Perl interface to `libexacct` to extract resource usage information from the extended accounting files. More information on the Perl interface to `libexacct` can be found in the *Solaris 10 Resource Manager Developer's Guide*.

The script processes the file(s) given on the command line and summarizes the CPU usage per project by selecting all task and task interval records in the file(s). Assuming that the extended accounting files conform to the `/var/adm/exacct/task<yymmdd>` naming convention, a monthly report for February 2005 can be generated by running the following script.

```
# cpureport.pl /var/adm/exacct/task0502*
PROJECT           USR+SYS
default                 0
group.dba               0
ora_mkt             76945
ora_sales          116620
system                342
user.root              59

# cat cpureport.pl
#!/usr/perl5/5.6.1/bin/perl
# cpureport.pl - extract CPU usage per project from extended
# accounting files (CPU time in seconds)
use strict;
use warnings;
use Sun::Solaris::Exacct qw(:EXACCT_ALL);
use Sun::Solaris::Project qw(:ALL);

my %proj = ();

die("Usage: $0 file [file ...]\n") unless ($#ARGV >= 0);

# Process all files given on the commandline
foreach my $arg (0 .. $#ARGV) {

  my $ef = ea_new_file($ARGV[$arg], &O_RDONLY) || die(ea_error_str());

  while (my $obj = $ef->get()) {
    if ( $obj->catalog()->id() == &EXD_GROUP_TASK ||
         $obj->catalog()->id() == &EXD_GROUP_TASK_INTERVAL ) {

      my $h = $obj->as_hash();  # returns all items in this group

      my $projid = $h->{EXD_TASK_PROJID};
      $proj{$projid}{CPU_SEC} += $h->{EXD_TASK_CPU_SYS_SEC};
      $proj{$projid}{CPU_NSEC} += $h->{EXD_TASK_CPU_SYS_NSEC};
      $proj{$projid}{CPU_SEC} += $h->{EXD_TASK_CPU_USER_SEC};
      $proj{$projid}{CPU_NSEC} += $h->{EXD_TASK_CPU_USER_NSEC};
    }
  }

  if (ea_error() != EXR_OK && ea_error() != EXR_EOF) {
    printf("\nERROR: %s\n", ea_error_str());
    exit(1);
  }
}
```

```
# Calculate total CPU time (usr + sys) and round to whole seconds
# and lookup project names (invent name if lookup fails).
for my $key ( keys %proj ) {
  my $one_second = 10 ** 9;     # ns per second

  if ( $proj{$key}{CPU_NSEC} >= $one_second ) {
    my $seconds = $proj{$key}{CPU_NSEC} / $one_second;
    $proj{$key}{CPU_SEC} += $seconds;

    if ( $proj{$key}{CPU_NSEC} % $one_second >= ($one_second / 2) ) {
      $proj{$key}{CPU_SEC}++;
    }
  }

  my $name = getprojbyid($key);
  if ( defined($name) ) {
    $proj{$key}{PROJECT} = $name;
  }
  else {
    $proj{$key}{PROJECT} = "<" . $key . ">";
  }
}

# Print the CPU usage for the projects sorted by project name
printf("PROJECT           USR+SYS\n");
for my $key ( sort { $proj{$a}{PROJECT} cmp $proj{$b}{PROJECT} } keys
%proj ) {
  printf("%-16s %8d\n", $proj{$key}{PROJECT}, $proj{$key}{CPU_SEC});
}

exit(0);
```

5

Dynamic Resource Pools

Some situations may be best served by partitioning available system resources, such as processors, into a number of discrete resource partitions. There are several reasons why this may be useful:

- Enforcing hard limits on the use of a resource. For instance, by creating a processor set and binding a process, project or zone to it, the CPU usage of the bound processes is effectively limited to the CPUs in the processor set. These processes cannot use processors outside of their set.
- Providing a guaranteed quantity of a resource. If an application requires a certain amount of CPU resources at all times, a processor set can be created for use by the application, thereby reserving the CPUs for application processes. Processes not bound to the set are unable to run on the processors in that set.
- Setting expectations. When deploying applications on a large server in phases, users may become accustomed to having fast response times as all resources are available to the application. As more applications are deployed, users may perceive a performance degradation. By partitioning the system so that the application received only the resources it needs, expectations can be set correctly from the start
- Partitioning by function, such as creating a partition for interactive users and a partition for batch jobs.

Processor Sets

The ability to partition a server using processor sets has been available since version 2.6 of the Solaris Operating System. Every system has at least one processor set, the system or default processor set that contains all of the processors in the system. Additional processor sets can be dynamically created and removed on a running system using the `psrset(1M)` command, provided that at least one CPU remains for the system processor set. Processes are bound to the default processor set by default, and can be bound to other processor sets on-the-fly. It is important to note that partitioning a system using processor sets may lead to

under utilization of the server since only processes bound to the processor set may use the processors in the set. If these processes do not use all of available CPU resources, the remaining CPU capacity in the set remains unused.

While processor sets are very useful, managing them can be a little cumbersome. System administrators must specify the physical CPU ID of the processor to add to a processor set. Since the physical ID of a CPU is hardware dependent, it varies between different hardware platforms, creating a close coupling between the processor set definition and the underlying hardware. Also, on systems that support Dynamic Reconfiguration, processors can be added and removed while the system is on-line. If a processor to be removed is used in a processor set, the system administrator must manually remove that processor from the set before the processor can be removed from the system. This requires the system administrator to have intimate knowledge of the configured processor sets and the hardware. Processor sets are referenced by a system generated ID, making it hard to remember what a specific set is used for, especially when multiple processor sets are present.

Resource Pools

The introduction of Resource Pools in the Solaris 9 OS significantly enhanced the ability to partition the system. Resource Pools provide a mechanism to create a persistent configuration of resource sets such as processor sets. The Resource Pools framework removes the link between the intention of the system administrator and the underlying hardware. Instead of creating a processor set by specifying physical CPU IDs, system administrators can now create a processor set with a chosen name by specifying the number of processors required, rather than their physical IDs. As a result, the definition of the processor set is no longer tied to a particular type of hardware.

System administrators can also specify a minimum and maximum number of processors for a set. The system assigns a number of processors between these values when creating the processor set on a specific system. This allows for more generic definitions that can be shared between systems. A configuration defining a set with at least one CPU and a maximum of three CPUs could be instantiated on a two-way system as well as on a larger server with more processors. Moving the definition to the larger server does not require any adjustment by the system administrator. The number of processors in the set on the larger server could be higher, depending on other processor sets defined in the system. The Resource Pools framework balances the number of processors in the set within the constraints set by the administrator.

On systems that support Dynamic Reconfiguration, the framework ensures that constraints are still met when removing processors from the system. If the total number of processors drops below the minimum number required for the active configuration, the Dynamic Reconfiguration operation is denied. If one of the processors being removed is part of a

processor set, the system reconfigures all processor sets in the system so that the processor is no longer in a set. Adding CPUs to a running system also causes a reconfiguration of processor sets, depending on the constraints set by the administrator.

Multiple configurations can be defined to adapt to changing resource requirements such as seasonal workloads or different daily and nightly workloads. The appropriate configuration can be instantiated by invoking the `pooladm(1M)` command manually or from a `cron(1M)` job.

Binding Processes To Pools

Instead of binding a process to a processor set directly, a process is bound to a Resource Pool using the `poolbind(1M)` command. A Resource Pool (or pool) is a logical collection of resource sets such as processor sets. While the processor set is the only type of resource set available in the Solaris OS, the resource pool abstraction allows other types of resource sets, such as memory sets, to be added in later Solaris OS versions.

A pool can optionally be associated with a scheduling class such as the Fair Share Scheduler (FSS) or the Real Time (RT) scheduling class. Processes bound to the pool are subject to that pool's scheduler, allowing the system to use different schedulers for different types of workloads. A server can be partitioned into two pools, one pool using the Fair Share Scheduler for applications, and a second pool using the Time Share scheduler (TS) for interactive users.

Multiple pools can be linked to the same resource set. As a result, it is possible to have a system with one processor set and several pools associated with the same processor set. This may not seem useful in a world with only processor sets. However, when other types of resource sets become available, it will be possible to let pools share a common processor set while giving each pool its own memory set, for instance.

The `poolbind(1M)` command allows administrators to bind processes, tasks, projects and zones to pools. A default pool binding for projects can be established by adding the `project.pool` attribute to the project entry in the project database. All processes started in the project are bound to the pool automatically. While the `project.pool` attribute designates only the default pool to bind to, specific processes in a project can still be bound to other pools if desired.

Fair Share Scheduler and Processor Sets

The previous discussion of the Fair Share Scheduler assumed all processors reside in the same processor set. When processor sets are present, the Fair Share Scheduler treats every processor set as a separate partition. CPU entitlement for a project is based on CPU usage in that processor set only. The CPU usage of a project in a processor set does not influence its entitlement in a different processor set. The Fair Share Scheduler calculates the proportion of CPU resources allocated to a project in a processor set by dividing the shares of the project by the number of shares of active projects in the processor set.

For example, consider a system with two processor sets, each containing one processor. Project A has two shares, and project B has one share. Both projects have enough processes to use all available CPU resources. Project B is the only one running in the first processor set. Since it is the only project in this set, project B is entitled to all CPU resources in the set. Both projects run in the second processor set. The number of active shares in this processor set is three (two from project A and one from project B). As a result, project A is entitled to $\frac{2}{3}$ of the processor set and project B is entitled to $\frac{1}{3}$. Project B's CPU use in the first processor set does not influence its entitlement in the second processor set.

Dynamic Resource Pools

In the Solaris 10 OS the Resource Pools facility has been further extended to provide automated resource allocation based on resource demands in the system and usage objectives set by the system administrator. This relieves system administrators from deciding how to optimally partition available resources for the current workload. Previously system administrators had to manually reassign resources to adapt to changing workloads. While fairly easy for relatively static workloads, this task may be challenging in an environment with highly variable resource demands.

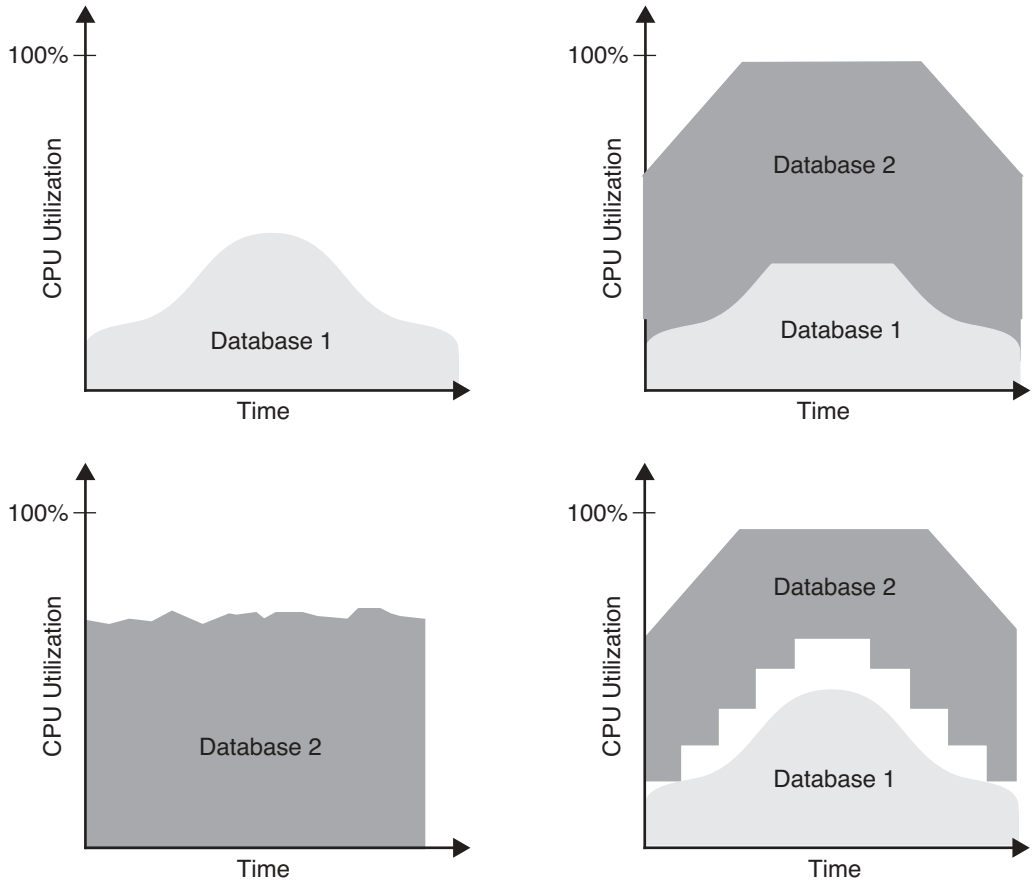

FIGURE 5-1 Dynamic resource pools lets the system adapt to changing workloads

Automated Resource Allocation

The Dynamic Resource Pools resource controller daemon `poold(1M)` is responsible for maintaining the resource allocation objectives set by system administrators. Toward this end, it creates an inventory of all available resources in the system. It continually monitors the active workloads in the system to determine if usage objectives can be met. If the resource controller detects that an objective is no longer being met, it evaluates possible alternative resource configurations to see if they can meet the objectives. If a viable alternative configuration exists, the resource controller reconfigures the resources accordingly. For processor sets, this is accomplished by moving processors between processor sets. If no alternative configuration exists that can meet objectives, no reconfiguration occurs. An appropriate message is logged, and the resource controller resumes workload monitoring.

Adding or removing resources using Dynamic Reconfiguration can also trigger a reconfiguration by the resource controller as the amount of available resource changes. Adding CPU capacity to a constrained system may create the opportunity for the resource controller to create a configuration that can meet objectives. Likewise, removing CPU capacity from the system may lead to the objectives no longer being met by the configuration.

Changes made to the objectives themselves by system administrators can also cause the resource controller to re-evaluate the configuration. The resource controller keeps a history of decisions made in the past, enabling it to rule out configuration changes that did not lead to improvement.

Even if the process of reconfiguration is automatic, system administrators can still directly manipulate the active configuration by transferring processors from one set to another. Note that doing so may or may not trigger actions by the resource controller.

Configuration Objectives

The resource controller offers several configuration objectives to influence decisions regarding possible resource configurations. These objectives can be combined and objectives can be assigned a precedence over each other. System administrators can choose from a number of different configuration objectives:

- **wt-load**
  This objective favors configurations that match resource allocations to resource demands. When this objective is in effect, a resource set that uses more resources is given more resources (within the minimum and maximum properties for the set).
- **locality**
  This objective is used to instruct the resource controller to take resource locality into consideration when allocating resources. On large servers such as the Sun Fire™ 15K server, the latency between resources on the same board and on a different board can vary. Depending on the application, latency may or may not be important. The locality objective allows the administrator to express the need for resource locality.
- **utilization**
  This objective favors configurations that allocate resources to partitions that are not meeting their utilization objective. System administrators can set target utilizations on the set using “less than”, “greater than” and “about” operators. The “less than” and “greater than” objectives can be combined to specify a target utilization range, such as between 50% and 80% utilization.

The configuration objectives are detailed in the `libpool(3LIB)` manual page. See Chapter 6 for an example of using objectives to control resource configuration.

Monitoring Resource Pools

System resource utilization can be monitored using the `poolstat(1M)` utility. This utility shows statistical data for every pool in the system. Data displayed includes the minimum, maximum and current size of the resource set, a measure of how much of the resource set is currently in use, as well as the load on the resource set.

The decisions made by the resource controller can be observed by consulting the `/var/log/pool/poold` log file .

Commands

The following commands are available to administer resource pools:

Command	Description
`pooladm(1M)`	Activate and deactivate the pools facility
`poolcfg(1M)`	Create and modify resource pool configuration files
`poold(1M)`	Monitors resource usage and adjusts resource allocation
`poolbind(1M)`	Bind processes, tasks, projects and zones to a pool
`poolstat(1M)`	Report active pool statistics

6

Resource Pools — An Example

This chapter presents an example of using Resource Pools to partition the available CPU resources on a system. Partitioning enables minimum and maximum amounts of CPU resources to be guaranteed to applications. Continuing with the sales and marketing database example presented earlier, assume the following policies. The sales database instance should always have at least 2 CPUs available to ensure a minimum level of service. Extra CPU capacity could increase service levels and the sales business unit is willing to pay extra for increased service levels. The marketing database requires at least one CPU, and a maximum of two CPUs, to achieve business objectives. The marketing business unit is not willing to be charged for more than two CPUs. These policies should require no manual intervention by the system administrator to adjust the number of CPUs in the processor sets.

Dynamic Resource Pools can be used to implement these requirements by creating a large processor set with at least two CPUs for the sales database, and a small processor set with at least one CPU and at most two CPUs for the marketing database. All remaining CPUs remain in the default processor set present on every system and which contains at least one CPU. When implemented on a system with six CPUs, the following configurations are possible:

Default Processor Set	Small Processor Set	Large Processor Set
2	2	2
1	2	3
1	1	4
2	1	3
3	1	2

The number of CPUs in each processor set can be dynamically adjusted to current system load according to allocation objectives set by the system administrator. For example, if high demand is experienced for CPU resources in the large processor set, the system might move processors from the small or default processor sets to the large processor set. When demand in the large processor set decreases, the system may move processors to the small or default processor sets.

Because the pools facility is disabled by default, pools must first be enabled using the `-e` (enable) option of the `pooladm(1M)` command. This creates a configuration with a processor set with all processors in the system and a default pool. The following output illustrates the configuration of a system with 6 CPUs after the `pooladm -e` command is run, and shows the default pool named `pool_default` and the default processor set `pset_default`.

```
# pooladm -e
# pooladm
system blondie
        string  system.comment
        int     system.version 1
        boolean system.bind-default true
        int     system.poold.pid 611

        pool pool_default
                int     pool.sys_id 0
                boolean pool.active true
                boolean pool.default true
                int     pool.importance 1
                string  pool.comment
                pset    pset_default

        pset pset_default
                int     pset.sys_id -1
                boolean pset.default true
                uint    pset.min 1
                uint    pset.max 65536
                string  pset.units population
                uint    pset.load 447
                uint    pset.size 6
                string  pset.comment

                cpu
                        int     cpu.sys_id 1
                        string  cpu.comment
                        string  cpu.status on-line

                cpu
                        int     cpu.sys_id 0
                        string  cpu.comment
                        string  cpu.status on-line

                cpu
                        int     cpu.sys_id 3
                        string  cpu.comment
                        string  cpu.status on-line

                cpu
                        int     cpu.sys_id 2
                        string  cpu.comment
                        string  cpu.status on-line

                cpu
                        int     cpu.sys_id 11
                        string  cpu.comment
                        string  cpu.status on-line

                cpu
                        int     cpu.sys_id 10
                        string  cpu.comment
                        string  cpu.status on-line
```

While the set currently contains six CPUs, the minimum (one) and maximum (65,536) number of CPUs are also set. Note the `system.bind-default`, `pool.default` and `pset.default` properties. These properties ensure that processes that do not bind to a specific pool are bound to the `pool.default` pool.

Note – This example assumes the users and projects created in Workload Management example still exist.

Creating a Pool

For the sales database, a processor set named `large` with at least two CPUs, and no upper bound on the number of CPUs, is created. Next, a processor set named `small` with at least one CPU and a maximum of two CPUs is created. A pool named `sales` is created and associated with the large processor set. A second pool named `marketing` is created and associated with the small processor set. Changes to the pools configuration can be made in two ways: to the active in-kernel configuration or to the `/etc/pooladm.conf` configuration file. The configuration contained in the `/etc/pooladm.conf` file can be instantiated by running the `pooladm -c` command. If desired, an alternate filename can be specified using the `-f` option. To save the currently active in-kernel configuration to a file, the `pooladm -s` command can be used. In this example, changes are made to the `/etc/pooladm.conf` configuration file, ensuring the changes persist across system reboots.

The initial configuration file is created from the running configuration, after which the processor sets and pools are added.

```
# poolcfg -c 'create pset large (uint pset.min=2;uint pset.max=65536)'
# poolcfg -c 'create pset small (uint pset.min=1;uint pset.max=2)'
# poolcfg -c 'create pool sales'
# poolcfg -c 'create pool marketing'
# poolcfg -c 'associate pool sales (pset large)'
# poolcfg -c 'associate pool marketing (pset small)'
```

These commands update the configuration contained in the `/etc/pooladm.conf` file, and have no effect on the active in-kernel configuration. This can be verified by displaying the active in-kernel configuration using the `poolcfg(1M)` command with the `-d` option.

Next, the configuration file is instantiated on the system. The processor set and the pool are created, and the system moves processors into the created processor set according to the available processors on the system and the `pset.min` and `pset.max` attributes of the configured processor sets. The in-kernel configuration now contains the following:

```
# pooladm -c
# poolcfg -dc info
system blondie
        string  system.comment
        int     system.version 1
        boolean system.bind-default true
        int     system.poold.pid 611

        pool marketing
                int     pool.sys_id 1
                boolean pool.active true
                boolean pool.default false
                int     pool.importance 1
                string  pool.comment
                pset    small
        pool pool_default
                int     pool.sys_id 0
                boolean pool.active true
                boolean pool.default true
                int     pool.importance 1
                string  pool.comment
                pset    pset_default
        pool sales
                int     pool.sys_id 2
                boolean pool.active true
                boolean pool.default false
                int     pool.importance 1
                string  pool.comment
                pset    large
        pset large
                int     pset.sys_id 1
                boolean pset.default false
                uint    pset.min 2
                uint    pset.max 65536
                string  pset.units population
                uint    pset.load 0
                uint    pset.size 2
                string  pset.comment
                cpu
                        int     cpu.sys_id 3
                        string  cpu.comment
                        string  cpu.status on-line
                cpu
                        int     cpu.sys_id 2
                        string  cpu.comment
                        string  cpu.status on-line
```

```
 pset small
                int     pset.sys_id 2
                boolean pset.default false
                uint    pset.min 1
                uint    pset.max 2
                string  pset.units population
                uint    pset.load 0
                uint    pset.size 2
                string  pset.comment

                cpu
                        int     cpu.sys_id 1
                        string  cpu.comment
                        string  cpu.status on-line

                cpu
                        int     cpu.sys_id 0
                        string  cpu.comment
                        string  cpu.status on-line

pset pset_default
                int     pset.sys_id -1
                boolean pset.default true
                uint    pset.min 1
                uint    pset.max 65536
                string  pset.units population
                uint    pset.load 4
                uint    pset.size 2
                string  pset.comment

                cpu
                        int     cpu.sys_id 11
                        string  cpu.comment
                        string  cpu.status on-line

                cpu
                        int     cpu.sys_id 10
                        string  cpu.comment
                        string  cpu.status on-line
```

Binding to a Pool

The sales database project is bound to the `sales` pool by adding the `project.pool` attribute to the project entry for the `ora_sales` project. Every new process started in this project is bound to the `sales` pool by default.

```
# projmod -sK "project.pool=sales" ora_sales
# projmod -sK "project.pool=marketing" ora_mkt
# cat /etc/project
system:0::::
user.root:1::::
noproject:2::::
default:3::::
group.staff:10::::
group.dba:100:Oracle DBA:::project.max-shm-memory=(privileged,2147483648,deny)
ora_mkt:101:Oracle Marketing:oracle::project.cpu-
shares=(privileged,10,none);project.max-shm-
memory=(privileged,2147483648,deny);project.pool=marketing
ora_sales:102:Oracle Sales:oracle::project.cpu-
shares=(privileged,20,none);project.max-shm-
memory=(privileged,2147483648,deny);project.pool=sales
```

Existing processes in the project are still bound to the default pool; they can be moved to the `sales` pool using the `poolbind(1M)` command. The following command binds all processes currently running in the project `ora_sales` to the `sales` pool. Start a new process in the `ora_sales` project to verify the pool binding.

```
# poolbind -p sales -i project ora_sales
# su - oracle
$ newtask -p ora_sales
$ id -p
uid=100(oracle) gid=100(dba) projid=100(ora_sales)
bash-2.05b
$ poolbind -q $$
1520    sales
```

Transferring CPUs

The system creates processor sets on a particular system based on the pool configuration and the number of CPUs in the system. In this example using a six CPU system, all three processor sets are created with two CPUs. The system administrator can manually move processors from one processor set to another to shrink or enlarge a processor set depending on the CPU requirements of applications. For example, end of month processing may require the `large` pool to contain four CPUs. The extra CPUs can be moved from the `small` and `default` processors sets using the `poolcfg(1M)` command:

```
# poolcfg -dc 'transfer 1 from pset pset_default to large'
# poolcfg -dc 'transfer 1 from pset small to large'
```

Adapting to Load

So far, the pool configuration is static. Changes in system load do not lead to configuration changes. The system administrator must manually move processors between sets to react to changes in utilization. By setting an objective, the system administrator tells the system to adapt the number of processors in a set to system demand. In this example, the utilization objective is used to ensure utilization of the `large` and `small` processor sets is kept below 75 percent to allow for spikes in the load.

```
# poolcfg -dc 'modify pset large (string pset.poold.objectives="utilization<75")'
# poolcfg -dc 'modify pset small (string pset.poold.objectives="utilization<75")''
# poolcfg -dc info
[...]
        pset large
                int     pset.sys_id 1
                boolean pset.default false
                uint    pset.min 2
                uint    pset.max 65536
                string  pset.units population
                uint    pset.load 182
                uint    pset.size 2
                string  pset.comment
                string  pset.poold.objectives utilization<75
[...]
```

Note – Until a patch for bug 6232648 is available, a workaround is needed for utilization objectives. Each processor set should have at least one 'pinned' CPU to prevent the issue described in the bug from occuring. The following command can be used to pin a CPU in a processor set. (Replace ID with the appropriate CPU ID.)

```
# poolcfg -dc 'modify cpu ID (boolean cpu.pinned=true)'
```

To see how the system adapts to varying demand for CPU resources, load is generated in the `small` processor set. It currently contains only one CPU since the second CPU was moved by the administrator in *Transferring CPUs* earlier. When the load exceeds the 75% utilization objective, the system attempts to move a processor from another processor set into the `small` processor set.

```
$ id -p
uid=100(oracle) gid=100(dba) projid=101(ora_mkt)
$ /usr/sbin/poolbind -q $$
666 marketing
$ nspin -n 4
```

The file `/var/log/pool/poold` can be observed to see the actions taken by the resource controller, such as moving a processor from one processor set to another:

```
Mar 22 15:28:33 Monitoring INFO: all evaluated objectives satisfied
Mar 22 15:28:48 Monitoring INFO: all evaluated objectives satisfied
Mar 22 15:29:03 Monitoring INFO: pset small utilization objective not satisfied (1,
utilization, '<', 75) with utilization 85.99 (control zone bounds exceeded)
Mar 22 15:29:03 Monitoring INFO: reconfiguration required
Mar 22 15:29:03 Optimization INFO: from pset large to pset small components [cpu 2]
not applied due to poor past results
Mar 22 15:29:03 Optimization INFO: applying move from pset large to pset small
components [cpu 1]
Mar 22 15:29:03 Configuration INFO: reconfiguring...
Mar 22 15:29:03 Configuration INFO: configuration complete
```

As shown the above output, the system decides to move processor 1 from the `large` processor set to the `small` processor set to satisfy utilization objectives. Stopping the load in the `small` processor set and adding load in the `large` processor set causes another reconfiguration after some time to satisfy the utilization objective on the `large` processor set.

This example only shows a tiny fraction of the possibilities enabled by Dynamic Resource Pools. More complex objectives can be created, such as combining different types of objectives and setting the importance of these objectives in relation to each other. See the `man` page for `libpool(3LIB)` for more information on setting objectives.

Saving the Dynamic Configuration

The last few changes have been made to the in-kernel configuration. To keep these changes across reboots, the in-kernel configuration must be saved to a file using the `pooladm -s` command. This command saves the configuration to the `/etc/pooladm.conf` file. The system automatically instantiates the configuration from this file at boot time.

7

Solaris Zones

Solaris Zones provide a means to create one or more virtual environments on a single operating system instance, shielding applications from details of the underlying hardware. Applications in a zone run in isolation from applications in other zones. They cannot see, monitor or affect processes running in another zone. Zones provide the following features:

- *Security*

  Network services can be run in a zone, limiting the damage that can be done to the system and other zones in case of a security violation. An intruder that is able to exploit a security hole in an application running in a zone can only do limited damage. The actions possible in a zone are restricted to subset of what is allowed in a normal system. For instance, it is not possible to load custom kernel modules, access kernel memory or create device nodes inside a zone.

- *Isolation*

  Applications requiring exclusive access to global resources, such as specific usernames or network ports, can run on the same machine using zones. Each zone has its own namespace, completely separate from other zones. Users in a zone are unable to monitor other zones, such as viewing network traffic or the activity of processes.

- *Virtualization*

  Zones present a virtualized environment to applications, removing the physical details of the hardware from view. This eases redeployment of applications on a different physical machine.

- *Granularity*

  Since zones are implemented in software, zones are not limited to granularity defined by hardware boundaries. Instead, zones offer sub-CPU granularity. Zones do not require dedicated CPU resources, dedicated I/O devices such as HBAs and NICs, or dedicated physical memory. As a result, even a system with a single processor can be used to host several zones.

- Transparency

  The environment presented to the application in a zone is nearly identical to the standard Solaris OS environment. There are no new, zone-specific APIs or ABIs to which applications must be ported. Some restrictions do exist due to security and isolation requirements. These restrictions mainly affect applications that perform privileged operations or need access to physical devices.

Zones Overview

The *global zone* encompasses the entire system and is comparable to a normal Solaris OS instance. It has access to the physical hardware and can see and control all processes. The administrator of the global zone can control the system as a whole. The global zone always exists, even when no other zones are configured. Inside the global zone are *local zones*. These zones are isolated from the physical hardware characteristics of the machine by the virtual platform layer. This layer provides the zones with a virtual network interface, one or more file systems and a virtual console.

Even though the virtual network interfaces may map to the same physical network interface, applications in different zones are prevented from seeing traffic from applications in other zones. Every zone has its own process environment and runs its own set of core Solaris OS services, including `inetd(1M)`, `syslogd(1M)`, `rpcbind(1M)`, and more. Applications running in a zone are unable to see applications running in other zones because of this private process environment. Zones are confined to their own subtree in the file system hierarchy and cannot access file systems of other zones or the global zone. All zones share the same operating system instance and therefore run the same Solaris OS version.

The virtual platform layer is not an emulation layer that translates requests from a zone into some other form or executes them on the zone's behalf. The role of the virtual platform layer is to instantiate and to connect virtualized resources to a zone. For instance, in the case of a virtual network interface, the virtual platform layer creates a logical interface on top of the physical network interface specified in the zone configuration. The IP address from the zone configuration is configured on the logical interface and it is made available to the zone.

One of the attributes of the logical interface is the zone in which it is configured. The kernel uses this attribute to virtualize the network interface by passing packets to the appropriate zone based on this attribute. A zone only sees packets that are destined for its own logical interfaces. Broadcast or multicast packets are replicated and sent to all zones as appropriate.

Virtualization of file systems in a zone is achieved via a restricted root similar to `chroot(2)`. A process in a zone is limited to files and file systems that can be accessed from the restricted root. Unlike chroot, a zone is not escapable. The virtual platform layer is responsible for creating the restricted root and mounting the file systems defined in the zone configuration on it.

Process isolation is accomplished by adding a reference to the zone to the process credentials. The kernel has been extended to use the zone ID as a means to restrict visibility of other processes. Only processes with the same zone ID are visible to a process in a zone. This selection is made *inside* the kernel and not available in utilities such as `ps(1)` or `kill(1)`, as that would make it possible to subvert the isolation by writing a `ps(1)` replacement.

As the zone ID is part of the credentials, the user ID namespace is also virtualized in zones. Every zone has its own user ID namespace. As a result, users in different zones with the same uid are in fact distinct users, even though they share the same numerical id. The virtualized user ID namespace also implies that passwords are unique to the zone.

The introduction of Least Privilege in the Solaris OS provides a set of fine-grained privileges to replace the concept of the omnipotent root user. Instead of performing checks against uid 0 to allow privileged operations, the kernel now checks for specific privileges required to perform privileged operations. In the past, it was sufficient to be the superuser to perform mount operations. Now, even the root user must have a specific privilege to perform mount operations. By restricting the privileges of root in the local zone to a set of privileges that are safe in a zone, the root user in a local zone can be given enough power to manage the zone without the ability to affect the system as a whole, such as rebooting the system. Restricting privileges by itself is not sufficient for isolation. Privileges only restrict the operations that can be performed, not the objects on which they are performed. This is accomplished by the isolation that zones provide.

It is therefore possible to delegate local zone administration to users by giving them access to the root account in a local zone. Since a user with uid 0 in one zone is different from a user with uid 0 in another zone, a local zone root user cannot compromise any other zone. However, the global zone root user should still be closely guarded as it has control over the system as a whole, and as such has access to all zones.

Administering Zones

Zone administration tasks can be divided into two parts, *global* zone administration tasks such as creating a zone, and *local* zone administration tasks such as performing configuration *within* a zone. The four primary global zone administration tasks are:

- *Configuration* — the global administrator defines the virtual platform properties, such as the required file systems and network interfaces
- *Installation* — the global administrator creates the zone on the system by creating and populating the part of the file system hierarchy reserved for the zone
- *Virtual Platform Management* — the global administrator uses zone tools to boot, halt or reboot the zone

- *Zone Login* — the global administrator can move in and out of the local zone from the global zone to assume the role of the local zone administrator

Zone Configuration

The first step of creating a zone on a system is defining its configuration using the `zonecfg(1M)` command. The configuration describes resources and properties that are required for the zone to operate:

- *Zone name* — A unique name for the zone. This name is only of interest in the global zone, it is distinct from the nodename of the zone. The name 'global' and names starting with 'SUNW' are reserved.
- *Zone path* — Every zone has a path to its root relative to the global zone's root directory. The zone's root path will be one level deeper; the 'root' directory under the zone path. To prevent unprivileged users in the global zone from traversing into the file system hierarchy of the zone, the zone path must be owned by root with mode 700. The zone 'root' directory should be owned by root and have mode 755 like a regular root directory.
- *File systems* — a zone may have file systems that should be mounted when the zone is booted
- *Network interfaces* — a zone may have one or more virtual network interfaces. The configuration specifies the IP address and the physical interface in the global zone on which it is to be created.
- *Devices* — a zone may require devices that need to be configured when the zone is booted. A default set of devices required in every zone is provided and can be imported into the configuration
- *Resource controls* — a zone may have resource controls that should be enabled when the zone is booted.
- *Properties* — a zone can have properties that control some aspect of the zone, such as the `autoboot` property. The `autoboot` property is comparable to the `auto-boot?` OBP parameter and determines if the zone is to be booted automatically when the global zone is booted.

Installing Zones

The zone configuration process is only concerned with the syntactic correctness of the configuration: it determines if the configuration could be created on *some* system, not necessarily *this* particular system. This is verified when the zone is installed on a system using the `zoneadm(1M) install` command. This command checks to see if all resources, such as the physical network interface specified in the configuration, are available. It then installs the files needed for the zone's root file system in the correct location under the zone

path, and creates the mount points for additional file systems specified in the configuration. When the installation is complete, the zone is ready to be booted. The root file system of a typical zone requires approximately 100 MB of disk space.

Virtual Platform Management

The virtual platform layer is implemented by the `zoneadmd(1M)` daemon. When the global administrator boots a zone using the `zoneadm boot` command, an instance of `zoneadmd` is created for that zone. The `zoneadmd` instance for the zone consults the zone configuration and creates a new zone. It then creates the virtual network interfaces, mounts the file systems, and creates the zone virtual console. Finally it starts an instance of `init(1M)` in the zone to further bringup the zone using SMF.

The global zone administrator can halt the zone using `zoneadm halt` and reboot the zone using `zoneadm reboot`. These commands do not perform an orderly shutdown when bringing down the zone. These operations can be considered the equivalent of the `setkeyswitch` operations on the system controller of larger Sun Fire systems. If an orderly shutdown of the zone is required, an `init 0` command should be done from inside the zone to ensure the stop methods of `smf(5)` services are executed, just like a domain on a Sun Fire server would be shutdown prior to running the `setkeyswitch off` command.

The `svc:/system/zones smf(5)` service in the global zone is used for zone startup and shutdown. All zones with the autoboot property set are started automatically when the global zone boots. When the global zone is stopped, the zones service performs an `init 0` in each running zone to do an orderly shutdown of services in the zone.

Zone Login

Since zones are implemented in software and require no dedicated hardware, the virtual platform provides a virtual console for each zone. The virtual console can be accessed from the global zone using the `zlogin(1M)` command with the `-C` option. The console for a zone cannot be shared by multiple users at the same time. The first user to connect to the console of a zone has exclusive access until disconnecting from the console. Access to the consoles of *other* zones is still possible.

When using the `zlogin` command in interactive, non-console mode an arbitrary number of `zlogin` sessions to the same zone may be open concurrently. Non-interactive `zlogin` is also possible, for example to allow the global administrator to perform scripted administration in zones. The use of zlogin requires a specific privilege.

Traditional remote login facilities such as `telnet(1)`, `rlogin(1)` and `ssh(1)` can still be used, provided they have not been disabled by the local zone administrator.

Commands

The following commands were added to the Solaris OS for zones:

Command	Description
`zonecfg(1M)`	Create, update and view zone configuration
`zoneadm(1M)`	Administer zones
`zlogin(1)`	Login to a zone from the global zone
`zonename(1)`	Print the name of the current zone
`zoneadmd(1M)`	Zones administration daemon

Several existing Solaris OS commands were enhanced to support zones:

Command	Description
`prstat(1M)`	`-Z`, `-z` options added
`ps(1)`	`-o` *zone*, `-o` *zoneid* options added
`pgrep(1)`	`-z` option added
`pkill(1)`	`-z` option added
`priocntl(1)`	`-i` option added
`renice(1)`	`-i` option added
`ipcs(1)`	`-Z`, `-z` option added
`ipcrm(1)`	`-z` option added
`ppriv(1)`	`-z` option added
`ifconfig(1M)`	`-Z` option added
`poolbind(1M)`	`-i` option added
`df(1)`	`-Z` option added
`coreadm(1M)`	`%z` token added

Local Zone Administration

Administering the local zone is largely the same as administering a normal Solaris-based system. Some of the administrative facilities are not available in the zone because they do not apply to zones, such as device configuration.

After the initial install of the zone by the global administrator, the system is in an unconfigured state. The normal sysid tools are run on the first zone boot to finish zone configuration. The prompts from these tools can be bypassed completely by creating a `/etc/sysidcfg` configuration file with all required parameters in the zone. Each zone runs its own core Solaris OS services, and therefore has its own name service. Consequently, one zone can use NIS, while another zone can use LDAP, DNS or local files for the name service.

File Systems

A number of file systems are mounted in the zone by the virtual platform layer when the zone is booted. In the default configuration, the following file systems are mounted in a zone:

- /, the zone root file system is mounted on `<zonepath>/root` in the global zone
- `/sbin`, `/usr`, `/lib` and `/platform` file systems are read-only loopback mounts from the global zone to enable text page sharing to reduce memory requirements. This also reduces the required disk footprint of the zone.
- `/dev` for the zone is mounted on `<zonepath>/dev` in the global zone
- `/proc`, `/dev/fd`, `/system/contract`, `/etc/svc/volatile`, `/etc/mnttab`, `/var/run` and `/tmp`

Additional file systems can be mounted in a zone if required, either by adding these file systems to the zone configuration using the `zonecfg(1M)` command, mounting them from within the zone through the local zone's `/etc/vfstab` file, mounting them by hand, or triggered by autofs.

UFS file systems to be mounted in the zone are configured using the `zonecfg(1M)` command by adding file system resources of type `ufs`. When the zone is booted, the system automatically mounts these file systems in the zone. This is the recommended and safest way to add UFS file systems to a zone.

Mounting UFS file systems from within the zone requires that the character and block device for the file system be explicitly exported to the zone by the global administrator. The file system can then be created by the local zone administrator. It is important to note that this

may expose the system as a whole to the risk of failure as the local administrator has access to the raw device and can possibly induce a panic by corrupting the UFS metadata on the disk. It is, therefore, not recommended to mount file systems in this manner.

Alternatively, the global administrator can create a UFS file system somewhere in the global zone and export this file system as a lofs mount to the zone. This prevents the local administrator from exposing the system to the risk described above.

NFS file systems can be mounted in the local zone, provided that they do not originate from the global zone, since it is not possible for a NFS server to mount its own file systems. See the `mount_nfs(1M)` manpage for more information. Therefore, zones can be NFS clients of any server except the global zone. Currently zones cannot be NFS servers.

Resource Management

The regular resource management facilities such as resource controls, projects, and more are available inside zones. Because projects are also virtualized inside a zone, each zone has its own project database. This allows a local zone administrator to configure projects and resource controls local to that zone.

Resource Pools

A zone can be bound to a Resource Pool through the `pool` property of the zone configuration. The zone is bound to the specified pool upon creation of the zone. The pool configuration can only be changed from the global zone. A zone cannot change the pool to which it is bound. Zones cannot span multiple pools, therefore, all processes in a zone run in the same pool. It is however possible to bind multiple zones to the same resource pool.

Extended Accounting

The Extended Accounting framework has been extended for Zones. Each zone has its own extended accounting files for task- and process-based accounting that contain accounting records exclusively for the zone. The extended accounting files in the global zone contain accounting records for the global zone and all local zones. Since the Extended Accounting framework uses a flexible and extensible format, the accounting records are now tagged with the zone name to which they apply. This allows the global zone administrator to extract per zone accounting data from the accounting files in the global zone.

Fair Share Scheduler and Zones

To prevent a local zone from monopolizing the system, the global zone administrator can set zone-wide resource controls. The `zone.cpu-shares` resource control limits the amount of CPU resources a zone is entitled to in the same way that the Fair Share Scheduler does this for projects. A zone with a higher number of `zone.cpu-shares` is allowed to use more CPU resources than a zone with a low number of shares. The `zone.cpu-shares` resource control is configured using the `zonecfg(1M)` command. Note that this requires the Fair Share Scheduler to be the default scheduler for the system, or that the zones be bound to a pool with a processor set with FSS as the scheduler.

Combined with the regular Fair Share Scheduler inside a zone, this leads to a two-level distribution of CPU resources. First, the `zone.cpu-shares` configured by the global zone administrator determine the amount of CPU resources to which a zone is entitled. The amount of CPU resources available to the zone is then further distributed across the active projects in the zone according to the `project.cpu-shares` defined by the local zone administrator.

Resource Controls

All standard resource controls are available inside the local zone and can be used by the local zone administrator to perform resource management in the zone. The global zone administrator can limit the number of lightweight processes (LWPs) created by a zone by setting the `zone.max-lwps` resource control on a zone.

8

Using Zones — An Example

The following example demonstrates the features provided by zones that facilitate consolidation. It shows how to run the two Oracle workloads from the *Managing Workloads* example on page 22 in a Solaris Container using zones. In that example, both workloads shared the same physical system as well as the file system namespace, name service, network port namespace, user and group namespaces, and more. The sharing of these namespaces can lead to undesirable and sometimes difficult to manage situations, such as when the databases are managed by two different DBA groups. The fact that there is only one `oracle` user requires close coordination between the DBA groups, since changes made to that user's environment by one DBA group may impact the other database instance. The same holds true for the sharing of the file system namespace, where a single `/var/opt/oratab` file is used by multiple Oracle instances.

Sharing namespaces can also inhibit the consolidation from a large number of servers onto fewer systems. Existing procedures and scripts may, for example, assume the system is dedicated to the application. Making changes to these procedures and scripts may be difficult, costly or even impossible.

Solaris Zones help resolve these issues because each zone is a virtualized environment with its own private namespaces that can be managed independently of other zones on the system. For instance, the `oracle` user in one zone is a completely different user from the `oracle` user in another zone — they can have different uids, passwords, login shells, home directories, etc. By running each Oracle instance in its own zone, the instances can be completely isolated from each other, simplifying their management. As far as the Oracle instance is concerned, it still runs on a dedicated system.

Requirements

Two zones each running their own Oracle instance are created. The zones require approximately 100 MB of disk space, and the Oracle software and a database each require about 4 GB of disk space.

Note – In this chapter, the prompt is set to the zone name to distinguish between the different zones.

Preparation

The Oracle instances for the sales and marketing databases are recreated in Zones in this example. Consequently, the existing instances created in Chapter 4 should be stopped and the associated user, projects and file system should be deleted. The pool configuration built in Chapter 6 should be disabled.

```
global # svcadm disable salesdb
global # svcadm disable mktdb
global # svccfg delete salesdb
global # svccfg delete mktdb
global # userdel -r oracle
global # projdel ora_sales
global # projdel ora_mkt
global # projdel group.dba
global # pooladm -x
global # pooladm -d
```

Creating the First Zone

The zone used for the marketing database is named `mkt`. To show how a file system is added to a zone, a separate file system is created on a SVM soft partition (`d200`). The file system may, of course, also be created on a standard disk slice. The virtual network interface for the zone with IP address 192.168.1.14 is configured on the physical interface `hme0` of the system. The directory for the zone is created in the global zone by the global zone administrator. The directory used for the zone must be owned by `root` and have mode `700` to prevent normal users in the global zone from accessing the zone's file system.

```
global # mkdir -p /export/zones/mkt
global # chmod 700 /export/zones/mkt
global # newfs /dev/md/rdsk/d200
```

Configuring the Zone

The zone is created based on the default template that defines resources used in a typical zone.

```
global # zonecfg -z mkt
mkt: No such zone configured
Use 'create' to begin configuring a new zone.
zonecfg:mkt> create
zonecfg:mkt> set zonepath=/export/zones/mkt
zonecfg:mkt> set autoboot=true
```

The virtual network interface with IP address 192.168.1.14 is configured on the `hme0` interface of the global zone.

```
zonecfg:mkt> add net
zonecfg:mkt:net> set address=192.168.1.14/24
zonecfg:mkt:net> set physical=hme0
zonecfg:mkt:net> end
```

The file system for the Oracle binaries and datafiles in the `mkt` zone is created on a soft partion named `d200` in the global zone. Add the following statements to the zone configuration to have the file system mounted in the zone automatically when the zone boots:

```
zonecfg:mkt> add fs
zonecfg:mkt:fs> type=ufs
zonecfg:mkt:fs> set type=ufs
zonecfg:mkt:fs> set special=/dev/md/dsk/d200
zonecfg:mkt:fs> set raw=/dev/md/rdsk/d200
zonecfg:mkt:fs> set dir=/u01
zonecfg:mkt:fs> end
zonecfg:mkt> verify
zonecfg:mkt> commit
zonecfg:mkt> exit
```

The zone configuration is now complete. The `verify` command verifies that the current configuration is syntactically correct. The `commit` command writes the in-memory configuration to stable storage.

Installing the Zone

The zone is now ready to be installed on the system.

```
global # zoneadm -z mkt install
Preparing to install zone <mkt>.
Checking <ufs> file system on device </dev/md/rdsk/d200> to be mounted
at </export/zones/mkt/root>
Creating list of files to copy from the global zone.
Copying <2584> files to the zone.
Initializing zone product registry.
Determining zone package initialization order.
Preparing to initialize <916> packages on the zone.
Initialized <916> packages on zone.
Zone <mkt> is initialized.
The file </export/zones/mkt/root/var/sadm/system/logs/install_log>
contains a log of the zone installation.
```

Booting the Zone

The zone can be booted with the `zoneadm boot` command. Since this is the first time the zone is booted after installation, the standard system identification questions must be answered, and are displayed on the zone's console. The console can be accessed from the global zone using the `zlogin(1M)` command.

```
global # zoneadm -z mkt boot
global # zlogin -C mkt
[Connected to zone 'mkt' console]

SunOS Release 5.10 Version Generic 64-bit
Copyright 1983-2005 Sun Microsystems, Inc.  All rights reserved.
Use is subject to license terms.
Hostname: mkt
Loading smf(5) service descriptions: 100/100
```

At this point, the normal system identification process for a freshly installed Solaris OS instance is started. The output of this process is omitted here for brevity, and the configuration questions concerning the name service, time zone, etc., should be answered as appropriate for the site. After system identification is complete and the root password is set, the zone is ready for use.

```
SunOS Release 5.10 Version Generic 64-bit
Copyright 1983-2005 Sun Microsystems, Inc.  All rights reserved.
Use is subject to license terms.
Hostname: mkt

mkt console login:
```

To disconnect from the console use `~.` (tilde dot) just like in `tip(1)`. The zone can now be accessed over the network using the `telnet(1)`, `rlogin(1)` or `ssh(1)` commands, just like a standard Solaris OS system. (Note that root can only login at the console unless the `/etc/default/login` file is updated).

```
mkt console login: root
Password:
Last login: Tue Mar 22 21:55:00 on console
Sun Microsystems Inc.   SunOS 5.10      Generic January 2005
# df -h
Filesystem             size   used  avail capacity  Mounted on
/                      7.9G   4.6G   3.2G    60%    /
/dev                   7.9G   4.6G   3.2G    60%    /dev
/lib                   7.9G   4.6G   3.2G    60%    /lib
/platform              7.9G   4.6G   3.2G    60%    /platform
/sbin                  7.9G   4.6G   3.2G    60%    /sbin
/u01                   7.9G   8.0M   7.8G     1%    /u01
/usr                   7.9G   4.6G   3.2G    60%    /usr
proc                     0K     0K     0K     0%    /proc
ctfs                     0K     0K     0K     0%    /system/contract
swap                    15G   272K    15G     1%    /etc/svc/volatile
mnttab                   0K     0K     0K     0%    /etc/mnttab
fd                       0K     0K     0K     0%    /dev/fd
swap                    15G     0K    15G     0%    /tmp
swap                    15G    24K    15G     1%    /var/run
```

The `/lib`, `/platform`, `/sbin`, and `/usr` file systems are read-only loopback mounts from the global zone. This reduces the required disk space for the zone considerably, and allows the sharing of text pages, leading to more efficient use of memory. These file systems appear in the zone because they are defined in the default template used to create this zone. All other

file systems are private to the zone. The /u01 file system is mounted in the zone during zone boot by zoneadmd. It is not mounted by the zone itself. Also note that the zone is unaware that the file system is in fact residing on /dev/md/dsk/d200.

Installing Oracle

The group dba and the user oracle are required to run the Oracle software. Since the Oracle software uses shared memory, and the maximum amount of shared memory is now a project resource control, a project is needed in which to run Oracle. The project ora_mkt project is created in the zone and the project.max-shm-memory is set to the required value (in this case 2 GB). Since the System V IPC parameters are resource controls in Solaris 10 OS, there is no need to update the /etc/system file and reboot.

```
mkt # mkdir -p /export/home
mkt # groupadd dba
mkt # useradd -g dba -d /export/home/oracle -m -s /bin/bash oracle
mkt # passwd oracle
mkt # projadd -c “Oracle” user.oracle
mkt # projadd -c "Oracle" -U oracle ora_mkt
mkt # projmod -sK "project.max-shm-memory=(privileged,2G,deny)" ora_mkt
mkt # cat /etc/project
system:0::::
user.root:1::::
noproject:2::::
default:3::::
group.staff:10::::
ora_mkt:101:Oracle:oracle::project.max-shm-memory=(privileged,2147483648,deny)
user.oracle:100:Oracle:::project.max-shm-memory=(privileged,2147483648,deny)
```

Note that the zone has its own namespace and that the user, group and project just created are therefore only visible inside the mkt zone.

The Oracle software and the database are installed in /u01. In this example, the Oracle software is installed in the zone itself to create an Oracle installation idependent from any other Oracle installations. The software could also be installed in the global zone and then loopback mounted in the local zones. This would allow sharing of the binaries by multiple zones, but also create a coupling between Oracle installations with regards to patch levels and more. This example shows how to use zones to consolidate Oracle instances with maximum isolation from each other, so in this case the software is not shared. The installation can now be performed as described on page 91. Since /usr is mounted read-only in the zone, the default location /usr/local/bin suggested by the Oracle Installer should be changed to a writable directory in the zone, such as /opt/local/bin. The marketing database can be created using the procedure on page 93.

Using the `smf` service for the marketing database from Chapter 4 (the *Managing Workloads* example) the database instance can be started by importing the manifest and enabling the `mktdb` service in the zone.

Creating the Second Zone

The first zone used a directory in `/export/zones` in the global zone. Since this does not limit the size of the root file system of the local zone it could fill up the file system in the global zone, where `/export/zones` is located. To prevent a local zone from creating this problem, the zone root file system is created on a separate file system. The second zone is for the sales database and requires the following resources:

- A 100 MB file system for the zone root file system mounted in the global zone on `/export/zones/sales`. This file system is created on a Solaris Volume Manager soft partition (`/dev/md/dsk/d100`). A normal slice could also be used but would be quite wasteful given the limited number of slices available on a disk.
- To show how devices can be used in a zone, the disk slice `c1t1d0s3` is exported to the zone by the global zone administrator. A UFS file system is created on this slice inside the zone. This requires that both the block and character devices for the slice be exported to the zone. Note that this is for demonstration purposes only and is not the recommended way to use UFS file systems in a zone.
- A virtual network interface with IP address 192.168.1.15 on the `hme0` interface of the global zone is also needed.

```
global # newfs /dev/md/rdsk/d100
global # mkdir -p /export/zones/sales
global # mount /dev/md/dsk/d100 /export/zones/sales
global # chmod 700 /export/zones/sales
```

Configuring and Installing the Second Zone

The steps required to configure and install this zone are the same as for the first zone, with the exception that two devices are added to the zone configuration.

```
global # zonecfg -z sales
sales: No such zone configured
Use 'create' to begin configuring a new zone.
zonecfg:sales> create
zonecfg:sales> set zonepath=/export/zones/sales
zonecfg:sales> set autoboot=true
zonecfg:sales> add net
zonecfg:sales:net> set physical=hme0
zonecfg:sales:net> set address=192.168.1.15/24
zonecfg:sales:net> end
zonecfg:sales> add device
zonecfg:sales:device> set match=/dev/rdsk/c1t1d0s3
zonecfg:sales:device> end
zonecfg:sales> add device
zonecfg:sales:device> set match=/dev/dsk/c1t1d0s3
zonecfg:sales:device> end
zonecfg:sales> verify
zonecfg:sales> commit
zonecfg:sales> exit
global # zoneadm -z sales install
Preparing to install zone <sales>.
Creating list of files to copy from the global zone.
Copying <2584> files to the zone.
Initializing zone product registry.
Determining zone package initialization order.
Preparing to initialize <916> packages on the zone.
Initialized <916> packages on zone.
Zone <sales> is initialized.
The file </export/zones/sales/root/var/sadm/system/logs/install_log>
contains a log of the zone installation.
```

Booting the Zone

The first time a zone is booted after installation, the system identification process is performed. It is possible to skip the system identification questions during the first boot of the zone by creating a `sysidcfg` file in the zone prior to the first boot. The location of the `sysidcfg` file from the global zone is `/export/zone/sales/root/etc/sysidcfg`. A sample `sysidcfg` file is shown below, and can be customized to fit the situation.

```
global # cat /export/zone/sales/root/etc/sysidcfg
system_locale=C
timezone=US/Pacific
network_interface=primary {
     hostname=hostname
terminal=xterm
security_policy=NONE
name_service=NIS {
     domain_name=yourdomain.com
}
root_password=sS3G0h84sqwJA
```

To suppress the question about the NFS version 4 domain, set the `NFSMAPID_DOMAIN` line in the `/export/zones/sales/root/etc/nfs/default` file to the appropriate value for your site and create the `/export/zones/sales/root/etc/.NFS4inst_state.domain` file.

The `/dev/dsk/c1t1d0s3` and `/dev/rdsk/c1t1d0s3` devices are added to the zone configuration to show how devices can be imported into a zone. Note that the only devices present in the `/dev/dsk` and `/dev/rdsk` directories are the devices that were explicitly added to the zone configuration.

```
global # zoneadm -z sales boot
global # zlogin sales
sales # ls -l /dev/dsk
total 0
brw-r-----   1 root     sys       32,  3 Mar 24 11:44 c1t1d0s3
sales # ls -l /dev/rdsk
total 0
crw-r-----   1 root     sys       32,  3 Mar 24 11:44 c1t1d0s3
```

A new file system is created and added to the zone's `/etc/vfstab` file.

```
sales # newfs /dev/rdsk/c1t1d0s3
sales # mkdir /u01
sales # mount /dev/dsk/c1t1d0s3 /u01
sales # cat /etc/vfstab
#device         device          mount           FS      fsck    mount   mount
#to mount       to fsck         point           type    pass    at boot options
#
/proc           -               /proc           proc    -       no      -
ctfs            -       /system/contract        ctfs    -       no      -
objfs           -       /system/object  objfs   -       no      -
fd              -               /dev/fd         fd      -       no      -
swap            -               /tmp            tmpfs   -       yes     -
/dev/dsk/c1t1d0s3 /dev/rdsk/c1t1d0s3 /u01       ufs     2       yes     -

sales # df -h
Filesystem             size   used  avail capacity  Mounted on
/                       94M    70M    14M    83%    /
/dev                    94M    70M    14M    83%    /dev
/lib                   7.9G   4.6G   3.2G    60%    /lib
/platform              7.9G   4.6G   3.2G    60%    /platform
/sbin                  7.9G   4.6G   3.2G    60%    /sbin
/usr                   7.9G   4.6G   3.2G    60%    /usr
proc                     0K     0K     0K     0%    /proc
ctfs                     0K     0K     0K     0%    /system/contract
swap                    15G   272K    15G     1%    /etc/svc/volatile
mnttab                   0K     0K     0K     0%    /etc/mnttab
fd                       0K     0K     0K     0%    /dev/fd
swap                    15G     0K    15G     0%    /tmp
swap                    15G    24K    15G     1%    /var/run
/dev/dsk/c1t1d0s3      4.9G   5.0M   4.9G     1%    /u01
```

Notice the difference beteen the `/u01` file system in this zone and the `/u01` file system in the `mkt` zone. In this zone the physical device is visible while in the `mkt` zone it is not visible.

Installing Oracle

The installation of the Oracle software is the same as that for the mkt zone. Since the zones have completely separate namespaces, the user, group and project for Oracle must be created in this zone also. The project `user.oracle` should have the resource control `project.max-shm-memory` added to it to allow Oracle access to the required shared memory.

```
sales # mkdir -p /export/home/oracle
sales # groupadd dba
sales # useradd -g dba -m -d /export/home/oracle -s /bin/bash oracle
sales # passwd oracle
sales # projadd -c "Oracle" -U oracle ora_sales
sales # projmod -sK "project.max-shm-memory=(privileged,2G,deny)" ora_sales
sales # cat /etc/project
system:0::::
user.root:1::::
noproject:2::::
default:3::::
group.staff:10::::
ora_sales:100:Oracle:oracle::project.max-shm-memory=(privileged,2147483648,deny)
```

The Oracle installation can now be performed as described page 91. Since `/usr` is mounted read-only from the global zone, the default location `/usr/local/bin` suggested by the Oracle Installer should be changed to a writable directory such as `/opt/local/bin`. The sales database can be created using the procedure on page 93. Using the `smf` service for the sales database from Chapter 4 (the *Managing Workloads* example), the database instance can be started by importing the manifest and enabling the `salesdb` service in the zone.

Controlling CPU Consumption of Zones

The `zone.cpu-shares` resource control can be used to limit the CPU usage of zones with respect to other zones. This resource control is set through the `zonecfg(1M)` command. To give the `sales` zone twice the amount of CPU resources as the `mkt` zone, the number of `zone.cpu-shares` of the `sales` zone is set to twice the number of `zone.cpu-shares` of the `mkt` zone:

```
global # zonecfg -z sales
zonecfg:sales> add rctl
zonecfg:sales:rctl> set name=zone.cpu-shares
zonecfg:sales:rctl> add value (priv=privileged,limit=20,action=none)
zonecfg:sales:rctl> end
zonecfg:sales> exit
global # zonecfg -z mkt
zonecfg:mkt> add rctl
zonecfg:mkt:rctl> set name=zone.cpu-shares
zonecfg:mkt:rctl> add value (priv=privileged,limit=10,action=none)
zonecfg:mkt:rctl> end
zonecfg:mkt> exit
```

The resource control is made active at the next zone boot. To set the `zone.cpu-shares` resource control on a running zone the `prctl(1)` command can be used.

```
global # prctl -n zone.cpu-shares -r -v 20 -i zone sales
global # prctl -n zone.cpu-shares -r -v 10 -i zone mkt
```

To observe processes, the `prstat(1M)` command has been enhanced for zones with the `-Z` and `-z` options. The following `prstat -Z` output from the global zone shows processes running in the global and local zones. The bottom of the output shows a summary line for every running zone. Both zones are running eight instances of the `nspin` utility to show how CPU usage is controlled by the `zone.cpu-shares` resource control when contention arises for CPU resources. As can be seen from the output, the sales zone is given twice the amount of CPU resources, even while both zones are requesting the same amount of CPU resources from the system.

```
global # prstat -Z
PID USERNAME  SIZE   RSS STATE  PRI NICE      TIME  CPU PROCESS/NLWP
 28848 root     1144K  680K cpu10   12    0   0:00:34 8.2% nspin/1
 28844 root     1144K  680K cpu2    13    0   0:00:33 8.0% nspin/1
 28845 root     1144K  680K run      9    0   0:00:33 8.0% nspin/1
 28846 root     1144K  680K cpu3     8    0   0:00:33 8.0% nspin/1
 28843 root     1144K  816K run     11    0   0:00:33 7.8% nspin/1
 28849 root     1144K  680K cpu0    13    0   0:00:32 7.7% nspin/1
 28847 root     1144K  680K run     12    0   0:00:32 7.6% nspin/1
 28850 root     1136K  672K cpu1    14    0   0:00:32 7.5% nspin/1
 28772 root     1144K  680K run      8    0   0:00:18 4.1% nspin/1
 28771 root     1144K  680K run      3    0   0:00:19 4.1% nspin/1
 28775 root     1136K  672K run     10    0   0:00:19 4.1% nspin/1
 28774 root     1144K  680K run      9    0   0:00:19 4.1% nspin/1
 28769 root     1144K  680K run      1    0   0:00:19 4.0% nspin/1
 28768 root     1144K  816K run     12    0   0:00:17 4.0% nspin/1
 28770 root     1144K  680K run     13    0   0:00:17 3.9% nspin/1
ZONEID    NPROC  SIZE   RSS MEMORY      TIME  CPU ZONE
     9       17   43M   30M   0.4%   0:04:30  63% sales
    10       35  105M   69M   0.8%   0:02:37  32% mkt
     0       50  219M  127M   1.5%   0:01:24 0.1% global

Total: 102 processes, 331 lwps, load averages: 10.89, 5.64, 3.09
```

To observe processes in one or more specific zones, the `prstat` command can be given a list of zones to observe with the `-z` option. The following output was taken while both zones were executing eight instances of the `nspin` command. Only eight of the sixteen `nspin` processes are shown here (those in the sales zone).

```
global # prstat -z sales -a
PID USERNAME  SIZE   RSS STATE  PRI NICE      TIME  CPU PROCESS/NLWP
 28845 root     1144K  680K run      7    0   0:01:39 8.5% nspin/1
 28850 root     1136K  672K run     12    0   0:01:38 8.3% nspin/1
 28846 root     1144K  680K run      7    0   0:01:38 8.3% nspin/1
 28849 root     1144K  680K run     14    0   0:01:38 8.2% nspin/1
 28844 root     1144K  680K cpu0    18    0   0:01:39 8.2% nspin/1
 28843 root     1144K  816K run     11    0   0:01:38 8.1% nspin/1
 28847 root     1144K  680K cpu3    18    0   0:01:37 8.0% nspin/1
 28848 root     1144K  680K cpu10   23    0   0:01:39 7.8% nspin/1
 28401 root       11M 8584K sleep   59    0   0:00:02 0.0% svc.startd/11
 28399 root     2200K 1456K sleep   59    0   0:00:00 0.0% init/1
 28496 root     1280K 1032K sleep   59    0   0:00:00 0.0% sh/1
 28507 root     3544K 2608K sleep   59    0   0:00:00 0.0% nscd/23
 28516 root     1248K  920K sleep   59    0   0:00:00 0.0% utmpd/1
 28388 root        0K    0K sleep   60    -   0:00:00 0.0% zsched/1
 28517 root     2072K 1344K sleep   59    0   0:00:00 0.0% ttymon/1
 NPROC USERNAME  SIZE   RSS MEMORY      TIME  CPU
    16 root       39M   29M   0.3%   0:13:14  65%
     1 daemon   3528K 1312K   0.0%   0:00:00 0.0%


Total: 17 processes, 60 lwps, load averages: 15.47, 9.33, 4.85
```

Controlling CPU Consumption Inside Zones

The `zone.cpu-shares` resource control determines the CPU consumption of the zone as a whole in relation to other active zones. CPU consumption *inside* a zone is controlled by the `project.cpu-shares` resource control. Since zones have their own project database, the CPU consumption inside the zone can be controlled by the local zone administrator. To demonstrate this capability, two projects are added to the project database in the sales zone. The CPU shares of the projects are set to 40 and 10, giving the first project four times more CPU resources than the second project. Each project runs four instances of the `nspin` utility.

```
sales # projadd -K "project.cpu-shares=(privileged,40,none)" -U root abc
sales # projadd -K "project.cpu-shares=(privileged,10,none)" -U root xyz
sales # cat /etc/project
system:0::::
user.root:1::::
noproject:2::::
default:3::::
group.staff:10::::
ora_sales:100:Oracle:oracle::project.max-shm-memory=(privileged,2147483648,deny)
abc:101::root::project.cpu-shares=(privileged,40,none)
xyz:102::root::project.cpu-shares=(privileged,10,none)

sales # newtask -p abc
sales # id -p
uid=0(root) gid=1(other) projid=(abc)
sales # nspin -n 4 &
29004
sales # newtask -p xyz
sales # id -p
uid = 0(root) gid=1(other) projid=(xyz)
sales # nspin -n 4 &
29008

sales # prstat -J
   PID USERNAME  SIZE   RSS STATE  PRI NICE      TIME  CPU PROCESS/NLWP
 29009 root     1144K  680K cpu11   17    0   0:02:19  13% nspin/1
 29008 root     1144K  680K run     22    0   0:02:16  13% nspin/1
...
 28507 root     3680K 2888K sleep   59    0   0:00:00 0.0% nscd/24
 28997 root     1280K 1032K sleep   59    0   0:00:00 0.0% sh/1
PROJID    NPROC  SIZE   RSS MEMORY      TIME  CPU PROJECT
   101        5 5808K 3832K   0.0%   0:09:09  52% abc
   102        5 5808K 3832K   0.0%   0:02:40  14% xyz
     1        5   13M   10M   0.1%   0:00:00 0.0% user.root
     0        8   33M   24M   0.3%   0:00:08 0.0% system
Total: 23 processes, 67 lwps, load averages: 15.89, 13.20, 11.70
```

```
global # prstat -Z
PID USERNAME  SIZE   RSS STATE  PRI NICE      TIME  CPU PROCESS/NLWP
 29009 root     1144K  680K cpu11   28    0   0:03:35  13% nspin/1
...
 29004 root     1144K  680K run     24    0   0:01:01 3.5% nspin/1
 29006 root     1136K  672K run     27    0   0:01:01 3.4% nspin/1
ZONEID    NPROC  SIZE   RSS MEMORY      TIME  CPU ZONE
     9       21   49M   36M   0.4%   0:18:17  65% sales
    10       35  105M   70M   0.8%   1:35:49  34% mkt
     0       54  244M  138M   1.7%   0:01:25 0.0% global

Total: 110 processes, 340 lwps, load averages: 15.98, 13.96, 12.13
```

In this case, with only the `sales` and the `mkt` zones active, the `sales` zone is entitled to the following percentage of available CPU resources, as calculated by:

$$\texttt{zone.cpu-shares}_{\texttt{sales}}\texttt{/ (zone.cpu-shares}_{\texttt{sales}}\texttt{ + zone.cpu-shares}_{\texttt{mkt}}\texttt{)= 20 / (20 + 10) * 100 = 66\%}$$

This 66% is then distributed among the projects in the zone. The project `abc` is entitled to the following percentage of available CPU resources:

$$\texttt{project.cpu-shares}_{\texttt{abc}}\texttt{ / (project.cpu-shares}_{\texttt{abc}}\texttt{ + project.cpu-shares}_{\texttt{xyz}}\texttt{) * 66\% = 40 / (40 + 10) * 66\% = 53\%}$$

The `xyz` project is entitled to 13 percent of total CPU resources (as calculated by 10 / (40 + 10) * 66% = 13%). The output from the `prstat -J` command in the sales zone confirms that this is the case. Note that the global zone has been omitted from the calculations for simplicity. It does, however, use some CPU resources so the numbers calculated may differ slightly from observed behavior.

9

Containers — An Example

Combining the Resource Management and Zones features available in Solaris 10 OS, a system administrator can create Solaris Containers tailored for a specific use. Building on the examples in the previous chapters, assume the administrator wants to run the following workloads on a single SMP system:

- The production sales database
- The production marketing database
- A development environment for the marketing database with multiple developers working on application development
- A development environment for the sales database
- System management

The following issues prevent successful consolidation onto a single system:

- The databases are managed by two different DBA organizations that each have their own (conflicting) standards
- The database systems use different naming services
- The development systems use the same usernames, file system paths and Oracle SIDs as the production environment
- The database instances should be guarateed a certain minimum and maximum amount of CPU capacity at all times
- The production systems should get preferential treatment over the development systems
- The sales database is the most important workload
- Developers should not be able to monopolize the CPU resources on the development systems
- The marketing department is willing to pay for a maximum of two CPUs

The problem is that the sales and marketing databases cannot co-exist on a single system because of different database administration standards and the use of different naming services. This can be overcome by using a separate zone for each workload. The issue of the development environments sharing naming with production can also be overcome with zones. Each zone has its own namespace for users, file systems, network ports and naming services.

The guarantee for minimum and maximum CPU capacity can be ensured by using Dynamic Resource Pools and the Fair Share Scheduler. Resource Pools provide hard upper and lower bounds at the granularity of a CPU. By creating a pool for the sales production database, a pool for the marketing database, and a pool for all other workloads, the production databases can be given guaranteed CPU capacity.

The demand for preferential treatment of the production systems can be implemented using the Fair Share Scheduler by assigning the production zones more `zone.cpu-shares` than development zones. When contention for CPU resources arises, the production zones are given more CPU resources than the other zones.

To prevent a developer from monopolizing the CPU resources in a development zone, each developer is assigned to a project with a certain amount of `project.cpu-shares`. The Fair Share Scheduler is used inside the development zones to distribute CPU resources according to the shares.

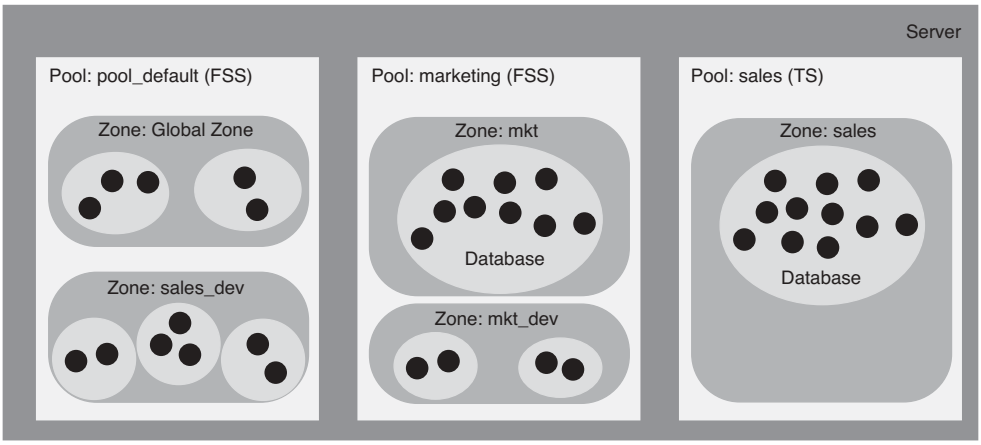

FIGURE 9-1 Zones and the resource management features of the Solaris 10 OS work together to enable applications to co-exist on systems

This leads to the following design:

- The resouce pool `sales` with a processor set `large` of at least two CPUs and no upper bound on CPUs
- Bound to this pool is a single zone `sales`, thus allowing exclusive access to the CPUs in the `large` processor set for the sales production database only
- Inside the zone there is a single project `ora_sales`, used to limit the amount of shared memory
- The pool uses the default Time Sharing (TS) scheduler since there is no need to arbitrate between zones in the pool or between projects in the zone
- A resource pool called `marketing` with a processor set `small` of at least one and at most two CPUs
- Bound to this pool are two zones, `mkt` for the marketing production database and `mkt_dev` for the development database

- This pool uses the Fair Share Scheduler (FSS) to arbitrate between the two zones bound to the pool and between projects in those zones
- Inside each zone is a single project `ora_mkt` to limit shared memory for the Oracle instance
- Each developer is assigned a unique project in the development zone with a suitable amount of CPU shares
- The default resource pool `pool_default` with a processor set with at least one CPU
- Bound to this pool are the `global` zone and the `sales_dev` zone for the sales database developers
- This pool uses the FSS scheduler to arbitrate between the two zones bound to the pool, and between projects in those zones
- Each developer is assigned a unique project in the development zone with a suitable amount of CPU shares

Container Construction

Creating the Pools

The pool configuration built in Chapter 6 almost matches the design, and could be used as a basis to create the required pools. However, the pools are created from scratch in order to show all relevant steps in a single location.

1. Enable the Resource Pools facility and save the default configuration to the `/etc/pooladm.conf` file. (The default configuration constists of a processor set `pset_default` with all CPUs and a single pool `pool_default`.)

```
global # pooladm -e
global # pooladm -s
```

2. Create the `sales` resource pool with TS as the scheduler and the `large` processor set with at least two CPUs.

```
global # poolcfg -c 'create pset large (uint pset.min=2; uint pset.max=65536)'
global # poolcfg -c 'create pool sales (string pool.scheduler="TS")'
global # poolcfg -c 'associate pool sales (pset large)'
```

3. Create the `marketing` resource pool with FSS as the scheduler and the small processor set with between one and two CPUs.

```
global # poolcfg -c 'create pset small (uint pset.min=1; uint pset.max=2)'
global # poolcfg -c 'create pool marketing (string pool.scheduler="FSS")'
global # poolcfg -c 'associate pool marketing (pset small)'
```

4. Set the scheduler for the default pool to the Fair Share Scheduler and instantiate the pool configuration just created:

```
global # poolcfg -c 'modify pool pool_default (string pool.scheduler="FSS")'
global # pooladm -c
global # poolcfg -dc info

system blondie
        string  system.comment
        int     system.version 1
        boolean system.bind-default true
        int     system.poold.pid 29072

        pool marketing
                int     pool.sys_id 5
                boolean pool.active true
                boolean pool.default false
                string  pool.scheduler FSS
                int     pool.importance 1
                string  pool.comment
                pset    small
        pool sales
                int     pool.sys_id 6
                boolean pool.active true
                boolean pool.default false
                string  pool.scheduler TS
                int     pool.importance 1
                string  pool.comment
                pset    large
        pool pool_default
                int     pool.sys_id 0
                boolean pool.active true
                boolean pool.default true
                string  pool.scheduler FSS
                int     pool.importance 1
                string  pool.comment
                pset    pset_default
         pset large
                int     pset.sys_id 1
                boolean pset.default false
                uint    pset.min 2
                uint    pset.max 65536
                string  pset.units population
                uint    pset.load 0
                uint    pset.size 2
                string  pset.comment
```

```
cpu
                int     cpu.sys_id 3
                string  cpu.comment
                string  cpu.status on-line
cpu
                int     cpu.sys_id 2
                string  cpu.comment
                string  cpu.status on-line
pset small
         int     pset.sys_id 2
         boolean pset.default false
         uint    pset.min 1
         uint    pset.max 2
         string  pset.units population
         uint    pset.load 0
         uint    pset.size 2
         string  pset.comment
          cpu
                 int     cpu.sys_id 1
                 string  cpu.comment
                 string  cpu.status on-line
         cpu
                 int     cpu.sys_id 0
                 string  cpu.comment
                 string  cpu.status on-line

 pset pset_default
         int     pset.sys_id -1
         boolean pset.default true
         uint    pset.min 1
         uint    pset.max 65536
         string  pset.units population
         uint    pset.load 17
         uint    pset.size 2
         string  pset.comment

         cpu
                 int     cpu.sys_id 11
                 string  cpu.comment
                 string  cpu.status on-line

         cpu
                 int     cpu.sys_id 10
                 string  cpu.comment
                 string  cpu.status on-line
```

Binding Zones to Pools

Currently all zones are bound to the default pool because the `pool` property of the created zones has not been set, resulting in the zones being bound to the pool with the `pool.default` attribute set to true. Setting the zone's pool property to the name of a resource pool binds that zone and all of its processes to that pool when the zone is booted. Note that since the sales zone is bound to a resource pool with the normal TS scheduler, the `zone.cpu-shares` resource control is no longer applicable and is therefore removed from the zone configuration.

```
global # zonecfg -z sales set pool=sales
global # zonecfg -z sales remove rctl name=zone.cpu-shares
global # zonecfg -z mkt set pool=marketing
```

To bind a running zone to a pool without rebooting the zone, the `poolbind(1M)` command can be used. This dynamically rebinds the zone and its processes to a pool until the next zone boot. To have this change persist accross zone reboots, the zone's property should be set as shown above.

```
global # poolbind -q `pgrep -z sales -x init`
28399   pool_default
global # poolbind -p sales -i zoneid sales
global # poolbind -q `pgrep -z sales -x init`
28399   sales
global # poolbind -p marketing -i zoneid mkt
global # poolbind -q `pgrep -z mkt -x init`
28545   marketing
```

The `poolbind -q `pgrep -z sales -x init`` command is used to ascertain to which zone the current pool is bound by querying the binding of the `init(1M)` process of that zone. As can been seen, the sales zone was bound to the pool `pool_default` and is now bound to the `sales` pool.

Creating Development Zones

The development environments for both databases get their own zones, enabing them to use the same user names, projects and file system paths as the production environments. The development zone for the sales database, `sales_dev`, is bound to the default pool and shares the pool with all processes of the global zone. To prevent the `sales_dev` zone from monopolizing CPU resources, its `zone.cpu-shares` is set to the same value as that of the

global zone. This gives both zones equal access to CPU resources. When the Fair Share Scheduler is active in a resource pool, it only looks at processes in that pool. The amount of shares for the `sales_dev` zone is only relevant in relation to those of the global zone.

```
global # zonecfg -z sales_dev
sales_dev: No such zone configured
Use 'create' to begin configuring a new zone.
zonecfg:sales_dev> create
zonecfg:sales_dev> set zonepath=/export/zones/sales_dev
zonecfg:sales_dev> set autoboot=true
zonecfg:sales_dev> set pool=pool_default
zonecfg:sales_dev> add rctl
zonecfg:sales_dev:rctl> set name=zone.cpu-shares
zonecfg:sales_dev:rctl> add value (priv=privileged,limit=1,action=none)
zonecfg:sales_dev:rctl> end
[...]
global # chmod 700 /export/zones/sales_dev
global # zoneadm -z sales_dev install
[...]
global # zoneadm -z sales_dev boot
```

The development environment of the marketing database uses the same pool as the zone for the marketing production database. The Fair Share Scheduler is used to give preferential access to the production zone. By setting the `zone.cpu-shares` of the `mkt` zone to 50, and the `zone.cpu-shares` of the `mkt_dev` zone to 10, the production database is granted five times as much CPU resources as the development database.

```
global # zonecfg -z mkt 'select rctl name=zone.cpu-shares; set
          value=(priv=privileged,limit=50,action=none);end'

global # zonecfg -z mkt_dev
mkt_dev: No such zone configured
Use 'create' to begin configuring a new zone.
zonecfg:mkt_dev> create
zonecfg:mkt_dev> set zonepath=/export/zones/mkt_dev
zonecfg:mkt_dev> set autoboot=true
zonecfg:mkt_dev> set pool=marketing
zonecfg:mkt_dev> add rctl
zonecfg:mkt_dev:rctl> set name=zone.cpu-shares
zonecfg:mkt_dev:rctl> add value (priv=privileged,limit=10,action=none)
zonecfg:mkt_dev:rctl> end
[...]
global # chmod 700 /export/zones/mkt_dev
global # zoneadm -z mkt_dev install
[...]
global # zoneadm -z mkt_dev boot
```

The pool bindings for the zones can be verified using the `poolbind -q pid` command on every zone's `init(1M)` process.

```
global # poolbind -q `pgrep -z sales_dev -x init`
6718    pool_default
global # poolbind -q `pgrep -z sales -x init`
28399   sales
global # poolbind -q `pgrep -z mkt -x init`
28545   marketing
global # poolbind -q `pgrep -z mkt_dev -x init`
6579    marketing
global # poolbind -q `pgrep -z global -x init`
1       pool_default
```

Creating Development Users and Projects

Once all zones are created, it is time to create users and projects inside the development zones and set the appropriate resource controls to implement the design. The Fair Share Scheduler is used to prevent the developers from consuming the CPU resources. In both zones three users and three projects are created.

User	Project	Resource Controls	Value
oracle	ora_mkt	project.max-shm-memory	2 GB
		project.cpu-shares	100
oracle	ora_sales	project.max-shm-memory	2 GB
		project.cpu-shares	100
dev1	user.dev1	project.cpu-shares	10
dev2	user.dev2	project.cpu-shares	10

```
global # zlogin mkt_dev
mkt_dev # mkdir -p /export/home
mkt_dev # groupadd dba
mkt_dev # useradd -g dba -m -d /export/home/oracle -s /bin/bash oracle
mkt_dev # projadd -U oracle ora_mkt
mkt_dev # projmod -sK "project.max-shm-memory=(privileged,2G,deny)" ora_mkt
mkt_dev # projmod -sK "project.cpu-shares=(privileged,100,none)" ora_mkt
mkt_dev # useradd -m -d /export/home/dev1 -s /bin/bash dev1
mkt_dev # useradd -m -d /export/home/dev2 -s /bin/bash dev2
mkt_dev # projadd user.dev1
mkt_dev # projadd user.dev2
mkt_dev # projmod -sK "project.cpu-shares=(privileged,10,none)" user.dev1
mkt_dev # projmod -sK "project.cpu-shares=(privileged,10,none)" user.dev2
[Oracle installation omitted for brevity...]
```

```
global # zlogin sales_dev
sales_dev # mkdir -p /export/home
sales_dev # groupadd dba
sales_dev # useradd -g dba -m -d /export/home/oracle -s /bin/bash oracle
sales_dev # projadd -U oracle ora_sales
sales_dev # projmod -sK "project.max-shm-memory=(privileged,2G,deny)" ora_sales
sales_dev # projmod -sK "project.cpu-shares=(privileged,100,none)" ora_sales
sales_dev # useradd -m -d /export/home/dev1 -s /bin/bash dev1
sales_dev # useradd -m -d /export/home/dev2 -s /bin/bash dev2
sales_dev # projadd user.dev1
sales_dev # projadd user.dev2
sales_dev # projmod -sK "project.cpu-shares=(privileged,10,none)" user.dev1
sales_dev # projmod -sK "project.cpu-shares=(privileged,10,none)" user.dev2
[Oracle installation omitted for brevity...]
```

Verifying the Configuration

The configuration just built can be verified using the following steps:

1. Start the `prstat -Z` command in the global zone to observe the CPU utilization of the zones.
2. Start the `poolstat -r pset 5` command in the global zone to observe utlization in the resource pools.

3. Create load using the `nspin -n 4` command in the `mkt` zone as the user `oracle` in the `ora_mkt` project. Note the CPU consumption of the `mkt` zone peaks around 33% since the marketing resource pool to which the zone is bound consists of two CPUs. The other CPUs are idle.

4. Add the same load in the `mkt_dev` zone. The combined CPU usage of the `mkt` and `mkt_dev` zones is approximately 33% since they share the same resource pool. The `mkt` zone receives approximately 27% and the `mkt_dev` zone about 6% because the `mkt` zone has five times more `zone.cpu-shares` than the `mkt_dev` zone.

5. Add the same load in the `sales` zone. The `sales` zone receives 33% since it is bound to the `sales` pool, which also has two CPUs. The CPU consumption of the `mkt` and `mkt_dev` zones is not impacted by the CPU usage of the `sales` zone.

6. Add load in the `sales_dev` zone. This zone is bound to the default pool. As a result, it is able to use all of the remaining CPU capacity since it is the only zone in that pool using CPU resources.

7. Add the same load in the global zone. The global zone is also bound to the default pool, and has the same amount of `zone.cpu-shares` as the `sales_dev` zone. The CPU usage of both zones is therefore equal, and approximately 16 percent. The resulting `prstat -z` command output looks as follows:

```
global # prstat -Z
   PID USERNAME  SIZE   RSS STATE  PRI NICE      TIME  CPU PROCESS/NLWP
  1987 100      1144K  672K cpu3    20    0   0:04:17 8.4% nspin/1
[...]
  2031 root     1144K  808K run      7    0   0:00:49 4.1% nspin/1
ZONEID    NPROC  SIZE   RSS MEMORY      TIME  CPU ZONE
     2       15   42M   30M   0.4%   0:17:17  33% sales
     1       33  104M   69M   0.8%   0:23:00  27% mkt
     0       65  388M  179M   2.2%   0:03:36  17% global
     3       33  105M   71M   0.8%   0:06:27  16% sales_dev
     4       33  103M   69M   0.8%   0:03:53 5.7% mkt_dev

Total: 179 processes, 586 lwps, load averages: 19.81, 14.84, 8.04
```

8. Add load in the `sales_dev` zone in the `user.dev1` and `user.dev2` projects. The total CPU usage of the `sales_dev` zone remains the same. However, in the zone the CPU should now be divided across the three projects according to the `project.cpu-shares` in the zone. Notice that a zone bound to a resource pool is only aware of the CPUs in the associated processor set. As a result, the `sales_dev` zone only knows about two CPUs, and the usage shown in the output of the `prstat` command is therefore based on two CPUs. That is why *inside* the zone the three projects seem to use 50%. (The other 50% is used by the global zone that is also bound to the same pool.) The `user.dev1` and `user.dev2` projects receive 10/120ths each of that 50% since they each have 10 `project.cpu-shares` and `ora_sales` has 100 `project.cpu-shares`.

```
sales_dev # prstat -J
   PID USERNAME  SIZE   RSS STATE  PRI NICE      TIME  CPU PROCESS/NLWP
  2016 oracle   1144K  672K run     35    0   0:24:48  10% nspin/1
  2088 dev1     1136K  704K run      1    0   0:00:03 1.4% nspin/1
[...]
  2113 dev2     1144K  848K run     27    0   0:00:02 1.2% nspin/1
PROJID    NPROC  SIZE   RSS MEMORY      TIME  CPU PROJECT
   100        5 7456K 4864K   0.1%   1:39:06  40% ora_sales
   101        5 7464K 5072K   0.1%   0:00:12 5.3% user.dev1
   102        5 7464K 5072K   0.1%   0:00:08 5.0% user.dev2
     1        5   11M 9336K   0.1%   0:00:00 0.2% user.root
     0       26   93M   63M   0.8%   0:00:14 0.0% system
     3        1 2904K 2064K   0.0%   0:00:00 0.0% default
Total: 47 processes, 132 lwps, load averages: 14.43, 10.28, 8.84
```

This example illustrates some of the ways that Solaris Containers technologies can be used to facilitate consolidation. It should be noted that not all features must be used at the same time. Depending on the circumstances, some Solaris Container technologies such as Resource Management, Resource Pools and Zones, can be mixed and matched to meet the specific needs for a consolidation project. In some cases, just using the Fair Share Scheduler may be sufficient to meet requirements, while in other cases Solaris Zones can be the key technology to a succesful consolidation.

10

Summary

As organizations build large-scale information systems to solve business problems, they are seeking new ways to protect technology investments while building scalable and highly available IT infrastructures that can adapt change. The facilities available in Solaris Containers can be used to help reduce IT infrastructure cost and improve end user service level management. With Solaris Containers, organizations can:

- Consolidate applications onto fewer servers
- Ensure applications remain isolated from one another
- Partition servers into several independent execution environments
- Manage the data center as a set of compute resources

About the Author

Menno is a Technical Specialist in Sun's Client Solutions organization in the Netherlands, where he assists Sun customers in implementing and optimizing their systems. His specialities include Solaris OS, high end servers, and resource mangement. Prior to joining Sun in 1999, Menno held several technical positions in mainframe and UNIX environments.

Acknowledgements

The author would like to recognize Joost Pronk van Hoogeveen for his contributions to this article.

Ordering Sun Documents

The SunDocs[SM] program provides more than 250 manuals from Sun Microsystems, Inc. Readers living in the United States, Canada, Europe, or Japan, can purchase documentation sets or individual manuals through this program.

Accessing Sun Documentation Online

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

To reference Sun BluePrints OnLine articles, visit the Sun BluePrints OnLine Web site at: `http://www.sun.com/blueprints/online.html`

References

Solaris 10 Resource Manager Developer's Guide. 817-1975-10. To access this book online, go to `http://docs.sun.com`.

System Administration Guide:Basic Administration. 817-1985. To access this book online, go to `http://docs.sun.com`.

System Administration Guide:Network Services. 816-4555. To access this book online, go to `http://docs.sun.com`.

System Administration Guide:Solaris Containers — Resource Management and Solaris Zones. 817-1592. To access this book online, go to `http://docs.sun.com`.

Solaris Containers: Server Virtualization and Manageability, Technical White Paper, Sun Microsystems, September, 2004. This white paper is available for download at `http://www.sun.com/software/solaris/whitepapers.xml`.

A

Solaris Container Features — Historical Summary

Feature	Available in Solaris Version
Solaris Resource Manager 1.x	Solaris 2.6 (unbundled product)
Projects	Solaris 8 10/01
Extended Accounting	Solaris 8 10/01
Fair Share Scheduler	Solaris 9 FCS
Resource Pools	Solaris 9 FCS
IPQoS	Solaris 9 9/02
Perl interface for libexacct	Solaris 9 4/03
Resource Capping Daemon	Solaris 9 12/03
Dynamic Resource Pools	Solaris 10 03/05
Zones	Solaris 10 03/05

B

Oracle Installation

This appendix contains the steps for installing Oracle 9i for use in the examples in this document. It only contains steps that are needed for demonstrating particular Solaris OS features mentioned in this paper. It should not be considered a guide to install Oracle on production systems.

The installation of Oracle consists of the following steps:

- Installing the Oracle software
- Updating the `.profile` of the `oracle` user
- Creating a script to start and stop the Oracle instances during system startup and shutdown

This procedure assumes that the following requirements have been met:

- The Oracle software and databases will be installed under `/u01`
- The available diskspace on `/u01` is approximately 3 GB for the Oracle software only, extra space is needed for the databases.
- The Oracle 9.2.0.1.0 installation media is mounted on `/mnt`

The target directory should be owned by the `oracle` user:

```
# chown -R oracle:dba /u01
```

The Oracle Installer GUI must be run as the `oracle` user and requires the `DISPLAY` variable to be set:

```
# su - oracle
$ cd /mnt
$ DISPLAY=<your display>; export DISPLAY
$ Disk1/runInstaller
```

Answer the following at the prompts:

```
"welcome"
select <next>
"Inventory Location"
base directory: /u01/app/oracle/oraInventory
select <next>
"UNIX Group Name"
select <next>
execute /tmp/oraintRoot.sh
select <Continue> when done
"File locations"
- destination name: Home1
- destination path: /u01/app/oracle/product/9.2.0.1.0
select <next>
"Available Products"
- Oracle9i Database
select <next>
"Installation Types"
Enterprise Edition
select <next>
"Database Configuration"
Software Only
select <next>
"Summary"
select <Install>
execute /u01/app/oracle/product/9.2.0.1.0/root.sh as user root
select <OK> when done
select <Exit>
```

After the Oracle Installer is finished the `.profile` of the `oracle` user must be updated so it looks like this:

```
 cd
$ cat .profile
ORACLE_HOME=/u01/app/oracle/product/9.2.0.1.0
PATH=$ORACLE_HOME/bin:/usr/ccs/bin:/usr/bin:/usr/openwin/bin:/etc
export ORACLE_HOME PATH
$ exit
$
```

C

Creating a Database

This appendix describes the procedure to create a simple Oracle database for the examples in this document. The following script creates the required directories, the `init.ora` file for the database and an SQL script to create the database. Save the script as `createdb.sh` in the home directory of the `oracle` user.

```
#!/bin/sh
mkdir -p /u01/oradata/$ORACLE_SID/redo
mkdir -p /u01/oradata/$ORACLE_SID/ctl
mkdir -p /u01/oradata/$ORACLE_SID/undo
mkdir -p /u01/oradata/$ORACLE_SID/sys
mkdir -p /u01/oradata/$ORACLE_SID/temp
mkdir -p /u01/app/oracle/admin/$ORACLE_SID/bdump
mkdir -p /u01/app/oracle/admin/$ORACLE_SID/udump
mkdir -p /u01/app/oracle/admin/$ORACLE_SID/cdump

cat > createdb.sql <<EOF
create database $ORACLE_SID
controlfile reuse
logfile '/u01/oradata/$ORACLE_SID/redo/redo_01.log' size 64M reuse,
 '/u01/oradata/$ORACLE_SID/redo/redo_02.log' size 64M reuse,
 '/u01/oradata/$ORACLE_SID/redo/redo_03.log' size 64M reuse
datafile '/u01/oradata/$ORACLE_SID/sys/system01.dbf' size 512M reuse
 extent management local
default temporary tablespace temp
tempfile '/u01/oradata/$ORACLE_SID/temp/temp01.dbf' size 512M reuse
undo tablespace undo01
datafile '/u01/oradata/$ORACLE_SID/undo/undo01.dbf' size 512M reuse
character set WE8ISO8859P15
;
EOF
```

```
cat > $ORACLE_HOME/dbs/init$ORACLE_SID.ora <<EOF
control_files                   = /u01/oradata/$ORACLE_SID/ctl/
control01.ctl
db_name                         = $ORACLE_SID
db_file_multiblock_read_count   = 8
db_block_size                   = 8192
db_block_buffers                = 5000
shared_pool_size                = 10000000
audit_trail                     = false
timed_statistics                = true
max_dump_file_size              = 10240
log_checkpoint_interval         = 10000
log_buffer                      = 163840
undo_management                 = auto
filesystemio_options            = setall
global_names                    = TRUE
background_dump_dest            = /u01/app/oracle/admin/$ORACLE_SID/
bdump
user_dump_dest                  = /u01/app/oracle/admin/$ORACLE_SID/
udump
core_dump_dest                  = /u01/app/oracle/admin/$ORACLE_SID/
cdump
compatible= 9.2.0.1.0
EOF
```

The script uses the current value of ORACLE_SID (the database identifier) to generate the appropriate files. So to create the Marketing database, set ORACLE_SID to MKT, and to create the Sales database set ORACLE_SID to SALES before executing `createdb.sh`.

```
$ ORACLE_SID=MKT (or ORACLE_SID=SALES to create the SALES database)
$ export ORACLE_SID
$ ./createdb.sh
$ sqlplus "/ as sysdba"
SQL> starup nomount
SQL> @createdb.sql
SQL> @?/rdbms/admin/catalog
SQL> @?/rdbms/admin/catproc
SQL> shutdown
SQL> exit
```

D

SMF Services for Oracle

This appendix describes the manifests and start and stop methods for the simple SMF services for the Oracle instances used in the example chapters.

```
# cat /u01/app/method/ora
#!/bin/sh
#
# Usage: ora ‘start’ | ‘stop’ db_id
#
ORACLE_SID=$2
ORACLE_HOME=/u01/app/oracle/product/9.2.0.1.0
export ORACLE_SID ORACLE_HOME

case “$1” in
                'start')
                $ORACLE_HOME/bin/sqlplus "/ as sysdba" <<START_EOF
startup
START_EOF
                ;;
                'stop')
                $ORACLE_HOME/bin/sqlplus "/ as sysdba" <<STOP_EOF
shutdown immediate
STOP_EOF
                ;;
esac
exit 0
```

The manifest `/var/svc/manifest/site/salesdb.xml` defines the SMF service for the sales database.

```
<?xml version="1.0"?>
<!DOCTYPE service_bundle SYSTEM "/usr/share/lib/xml/dtd/service_bundle.dtd.1">
<service_bundle type='manifest' name='SalesDB'>
<service
        name='site/salesdb'
        type='service'
        version='1'>
        <single_instance />

        <dependency
                name='local-fs'
                type='service'
                grouping='require_all'
                restart_on='none'>
                <service_fmri value='svc:/system/filesystem/local' />
        </dependency>

        <exec_method
                type='method'
                name='start'
                exec='/u01/app/method/ora start SALES'
                timeout_seconds='0'>
                <method_context
                        project='ora_sales'>
                        <method_credential user='oracle' />
                </method_context>
        </exec_method>

        <exec_method
                type='method'
                name='stop'
                exec='/u01/app/method/ora stop SALES'
                timeout_seconds='0'>
                <method_context
                        project='ora_sales'>
                        <method_credential user='oracle' />
                </method_context>
        </exec_method>
        <instance name='default' enabled='false' />
</service>
</service_bundle>
```

The manifest `/var/svc/manifest/site/mktdb.xml` defines the SMF service for the marketing database.

```
<?xml version="1.0"?>
<!DOCTYPE service_bundle SYSTEM "/usr/share/lib/xml/dtd/service_bundle.dtd.1">
<service_bundle type='manifest' name='MarketingDB'>
<service
        name='site/mktdb'
        type='service'
        version='1'>

        <single_instance />

        <dependency
                name='local-fs'
                type='service'
                grouping='require_all'
                restart_on='none'>
                <service_fmri value='svc:/system/filesystem/local' />
        </dependency>

        <exec_method
                type='method'
                name='start'
                exec='/u01/app/method/ora start MKT'
                timeout_seconds='0'>
                <method_context
                        project='ora_mkt'>
                        <method_credential user='oracle' />
                </method_context>
        </exec_method>

        <exec_method
                type='method'
                name='stop'
                exec='/u01/app/method/ora stop MKT'
                timeout_seconds='0'>
                <method_context
                        project='ora_mkt'>
                        <method_credential user='oracle' />
                </method_context>
        </exec_method>

        <instance name='default' enabled='false' />
</service>
</service_bundle>
```