3

PARAM 9000 Operating Environment

3.1 Introduction

C-DAC initiated its PARAM supercomputer and PARAS operating environment project around 1989. C-DAC's decision to implement its own operating system for PARAM family of supercomputers was essential since, existing systems such as UNIX were not suitable for MPP systems. A new operating system kernel called PARAS microkernel splits the traditional kernel into two distinct parts; that would interact in a client-server model:

- **PARAS Microkernel**
- **Operating System Servers**

These two components put together are popularly called as the *PARAS Operating Environment*. The PARAS microkernel operates the underlying architecture and the operating system servers interacts both with users and the microkernel through which it accesses hardware resources such as disks, I/O devices, network connectivity, etc.

The PARAM 9000 Operating Environment seamlessly blends industry standards with parallel programming enhancements to provide both ease-of-use and high performance. It supports the Solaris 2.x operating system and custom built PARAS Parallel Programming Environment, with an optimized microkernel.

The PARAM 9000 can be configured to have one of two fundamental logical personalities; first the cluster personality and second the MPP personality. In the cluster personality all nodes of the system are configured with the industry standard Solaris OS from SunSoft. In the alternate personality, few nodes of the system run Solaris OS while the others are configured with PARAS microkernel and system servers. The use of a custom microkernel provides the required flexibility to extract maximum performance from the system.

The PARAM 9000 hardware architecture allows the nodes to be configured as compute nodes or service nodes. The compute nodes can only execute user jobs, while the service nodes in addition provide various services required to support the execution of user jobs. The services provided include file services, gateway services for external connectivity, etc. The service nodes provide gateway services (gateway nodes) to support external connectivity through LAN(Ethernet, FDDI, token ring).

3.2 PARAM 9000 as a Cluster

The system exports the Solaris 2.x operating system on all the nodes (Figure 3.1) and the PARAM network supports a Data Link Provider Interface (DLPI) driver which implements ISO Data Link Standard Definition (ISO 8886) and Logical Link Service Definition (ISO 8802.2). The DLPI support allows mapping of common network protocols such as TCP/IP [1]. With the support for Solaris and TCP/IP over the Param network the system provides an environment which is functionally equivalent to a

cluster of Solaris workstations over a LAN. Much of the software support available under a distributed Solaris environment becomes readily available on PARAM also. It includes system management, job management, storage management, databases, and message-passing libraries. Also, with PARAM 9000 providing workstation environments at the underlying node level, shrink-wrapped Solaris SPARC binaries run straightaway on PARAM 9000 nodes.

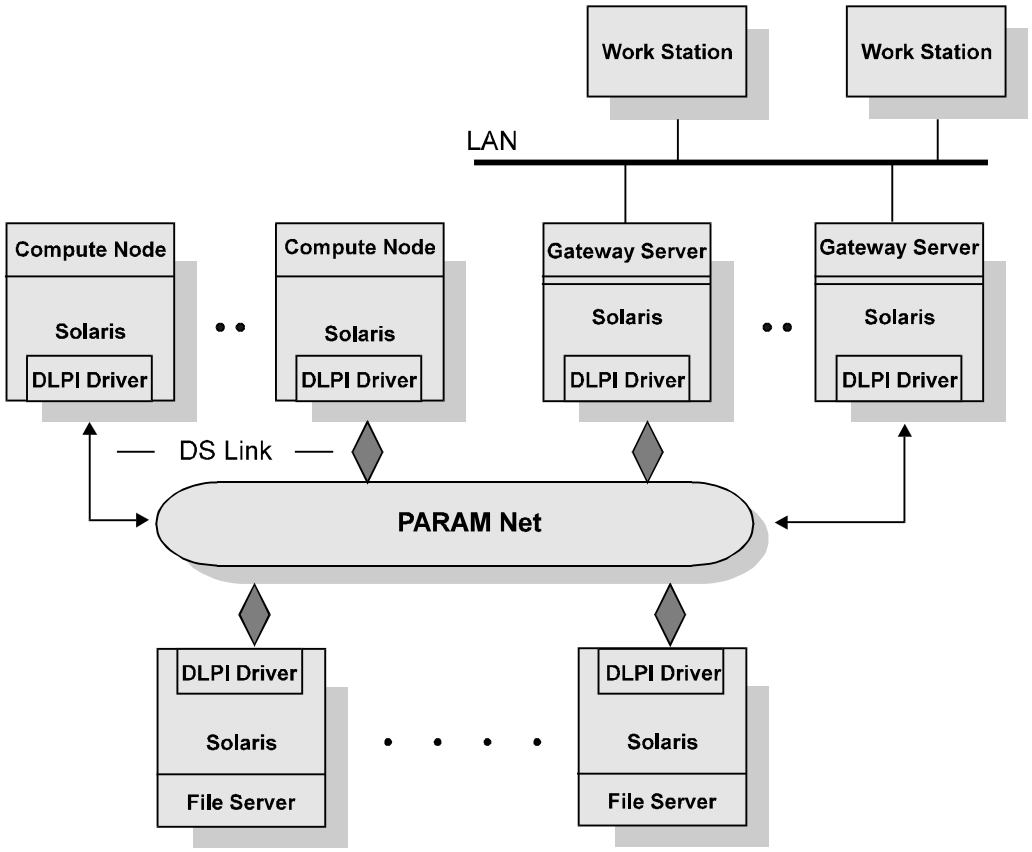


Figure 3.1: PARAM 9000 as a Cluster

3.2.1 Solaris

The Solaris Operating System version 2.x on PARAM 9000 is from SunSoft. This System V-style Unix provides a familiar development environment to the users. It has (Figure 3.2) robust networking support including a full and fast TCP/IP protocol stack and layered features such as RPCs (remote procedure calls), and the NFS (Network File System) distributed file system. It has an advanced programming environment, with ANSI-compliant optimizing C and C++ compilers, and tools to profile execution, and debug multithreaded programs. It also conforms to the following standards: SPARC ABI (Application Binary Interface), System V Release 4 (SVR 4), X/Open Portability Guide (XPG3), POSIX P1003.1, and X11R5.

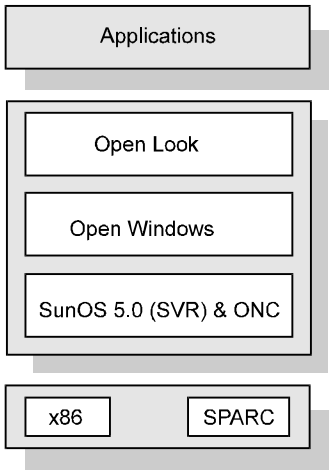


Figure 3.2: Solaris Components

The Solaris kernel supports multithreading and multiprocessing (Figure 3.3). It supports real-time scheduling features which are critical for multimedia. The kernel is dynamically loading and sizing. The kernel scheduler supports three classes of scheduling : *real-time*, *kernel*, and *time-sharing*. The priorities of threads in real-time and kernel classes do not change over time, but the priorities of threads in the time-sharing class do. The soft real-time support provided by the kernel ensures that real-time threads are given priority over other threads for the entire time they execute.

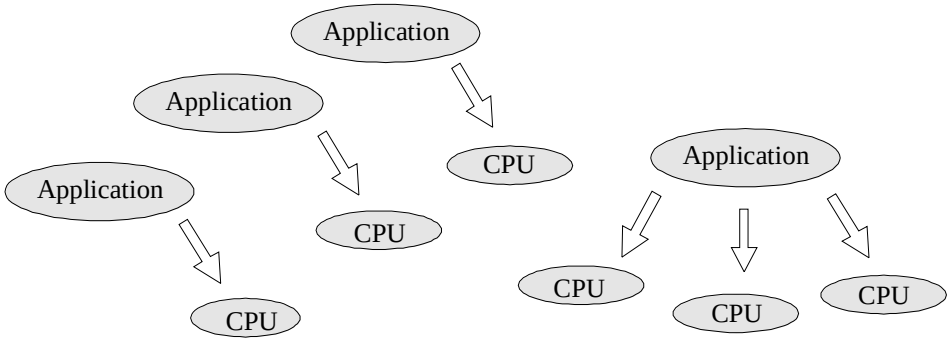


Figure 3.3: Multi-Tasking and Multi-Threading

The multithreading feature of the kernel includes support for two kinds of threads: *LWPs(lightweight processes)* and *user level threads*. The threads are intended to be sufficiently lightweight so that there can be thousands present and that synchronization and context-switching can be accomplished rapidly without entering the kernel. Threads are implemented using LWPs by the library. The LWPs are the kernel supported threads of control, scheduled by the kernel and, therefore consume kernel resources.

This two-level architecture allows the programmer to separate logical (program) concurrency from the required real concurrency, which is relatively costly, and to control both within a single programming model.

Solaris, in addition to BSD file system, also supports several types of non-BSD file systems to increase performance and ease of use. For performance there are three new file system types: CacheFS, AutoClient and TmpFS. The CacheFS caching file system allows a local disk to be used as an operating system managed cache of either remote NFS disk or CD-ROM file systems. With AutoClient and CacheFS, an entire local disk can be used as cache. The TmpFS temporary file system uses main memory to contain a file system. In addition there are other file systems like Proc file system and Volume file system to improve system usability.

Solaris supports distributed computing with support for storing and retrieving distributed information to describe the system and users through the NIS (Network Information Service) and NIS+ databases.

The Solaris GUI, OpenWindows, is a combination of X11R5 and the Adobe Post-script system which allows applications to be run on remote systems with the display shown along with local applications.

3.2.2 Job Management

Since PARAM 9000 can be used in a variety of environments it supports different job types: serial, parallel, interactive and batch. It can be configured to provide throughput or response time depending on the requirements of the job. PARAM 9000 supports job management through the CODINE job scheduling software.

CODINE (Computing in Distributed Networked Environments)

CODINE helps in optimal usage of compute resources of networked environments like PARAM 9000. It supports a convenient and easy to use graphical user interface. It provides dynamic and static load balancing, check-pointing and supports batch, interactive and parallel jobs. The GUI is based on user friendly motif interface. The following are the key features of CODINE:

- Support for different types of jobs
- Intelligent load balancing
- User defined check-pointing
- Detailed Statistics
- OSF/Motif GUI
- POSIX compliance
- Support for DCE

Job Scheduling in CODINE

The jobs are dispatched in accordance to their resource requirements. When required they are spooled and maintained in job queues. The jobs in the queue are scheduled based on the following algorithms: The queue, which is hosted by the least loaded host at the particular time is chosen. Queues are selected according to a given order defined by administrator. However overriding rules are provided for the administrator.

Parallel Job Execution

In order to execute a parallel job CODINE performs a selection of an appropriate sub-cluster. The user specifies the number of required queues via a generic request. Based on this information CODINE selects an appropriate set of queues from the currently available ones using the load level or the fixed ordered selection scheme. CODINE then starts up tasks and sets up other requirements of the respective parallel environments. In the context of the execution of parallel jobs the term “queue” is almost equivalent to machines or nodes, as it not efficient to run multiple tasks of a parallel program on queues hosted on the same machine.

User Interface (MOTIF GUI)

An OSF/Motif interface provides full interactive access to all user and administrative commands of CODINE which includes operations like queue configuration and management, job submission and their management. The statistical screen can be restricted to arbitrary time intervals, queue architectures and groups, including wall clock and CPU time utilization.

3.2.3 Communication Software

The main parallel programming model supported on PARAM 9000 is message-passing. In message-passing, the processes in a parallel application have their own private address space. They share data through messages and achieve synchronization implicitly through the act of sending and receiving messages. The programs are generally written with a Single-Program, Multiple-Data (SPMD) structure, where the same basic code executes against partitioned data. The processes exchange data through a message-passing library callable from standard languages. On PARAM 9000 both Parallel Virtual Machine (PVM) and Message Passing Interface (MPI) are supported (Figure 3.4). MPI is the emerging standard for message-passing while PVM is a popular interface for cluster environment. Both PVM and MPI are public domain implementations layered over the TCP/IP network protocol. The required interface for IP is provided by the DLPI driver for PARAM network.

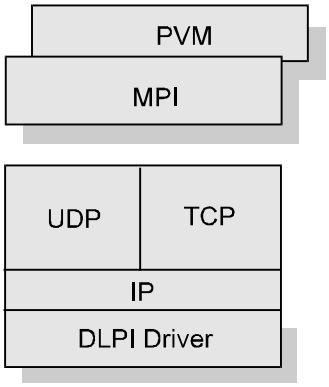


Figure 3.4: Communication Subsystem in Cluster mode

Parallel Virtual Machine (PVM)

PVM is a software system that enables a collection of heterogeneous computers to be used as a coherent and flexible concurrent computational resource. User programs written in C or Fortran, are provided access to PVM through the use of calls to PVM library routines for functions such as process

initiation, message transmission and reception, and synchronization via barriers or rendezvous. Users may optionally control the execution location of specific application components; the PVM system transparently handles the message routing, data conversion for incompatible architectures, and other tasks that are necessary for operation in a heterogeneous, network environment. PVM is ideally suited for concurrent applications composed of many interrelated parts. PVM supports heterogeneity at the application, machine, and network level. In other words, PVM allows application tasks to exploit the architecture best suited to their solution.

The PVM system has been used for a number of applications such as molecular dynamics simulations, superconductivity studies, and in the classroom as the basis for teaching concurrent computing. Xab and HeNCE are popular tools available with PVM. Xab generates traces of PVM messages, which can interface with the popular Paragraph tool for generating graphic displays. HeNCE allows applications to be developed at task-graph level.

Message Passing Interface (MPI)

MPI is the proposed standard for message passing for MIMD distributed memory concurrent computers. This standard is the result of meetings, extensive email discussions, and drafts, with participation from over 40 computer vendors and research organizations. MPI has adapted features from current message passing systems such as PVM, NX/2, PARMACS, Express, and P4. Aimed towards portability and efficiency, this interface provides the general as well as advanced user with rich functionality to write message passing parallel programs. With explicit support for the application programmer, library developer and tool developer, MPI is gaining wide acceptance amongst the parallel programming community.

The salient features of MPI include: unique system supplied tag for complete message security, functions to define groups and subgroups of tasks, point to point communication functions, blocking as well as non-blocking communication, collective communication, and application oriented process topologies. MPI has the required bindings for both FORTRAN 77 and C.

3.3 PARAM 9000 as a MPP

The PARAM 9000 OS architecture allows an optimized microkernel to be loaded on the compute nodes instead of Solaris (Figure 3.5). The communication subsystem over the microkernel has been highly optimized providing an order of magnitude improvement in performance.

3.3.1 The PARAS Microkernel

The PARAS Microkernel is a message based operating system kernel designed for massively parallel systems. It supports multiple tasks with a paged virtual memory space, multiple threads of execution within each task and message based inter-process communication (Figure 3.6). The basic abstractions implemented and managed by the microkernel are:

- task is the unit of resource allocation and memory address space
- thread is the unit of sequential execution
- message is a stream of bytes used as the unit of communication
- port is the unit of addressing for point-to-point communication of messages
- port group is the unit of addressing for one-to-many communication of messages
- region which is the unit of virtual memory usage

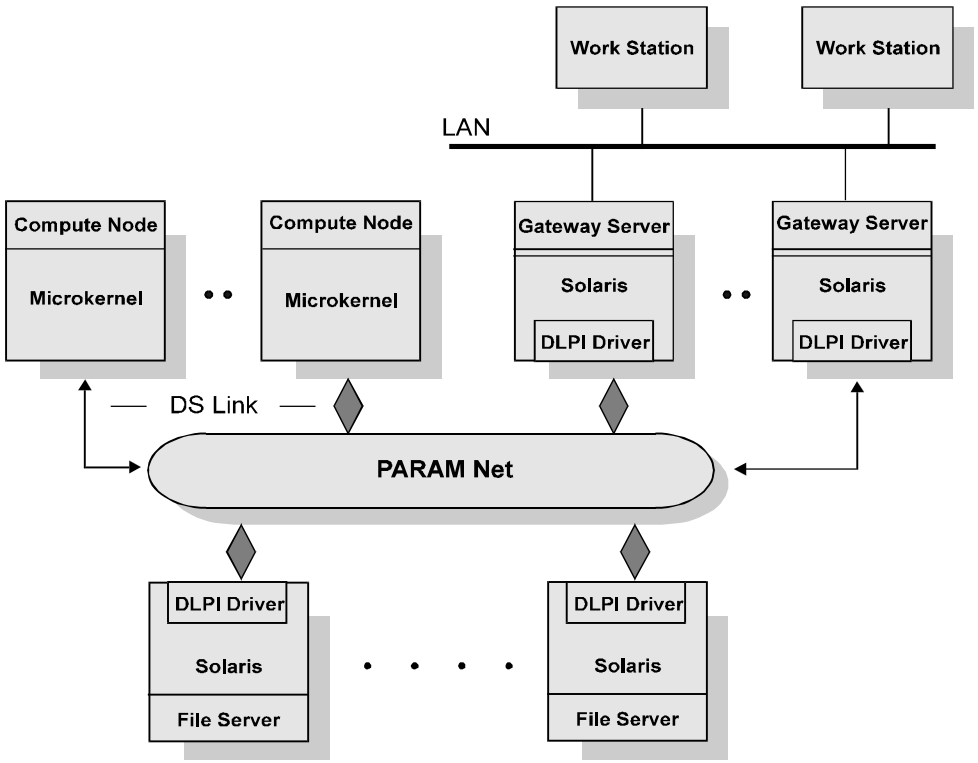


Figure 3.5: PARAM 9000 as MPP

Message passing is the primary means of communication both among tasks, and to some extent between tasks and the kernel itself. Location independent inter-process communication is supported through the port abstraction with support for both synchronous and asynchronous modes of communication.

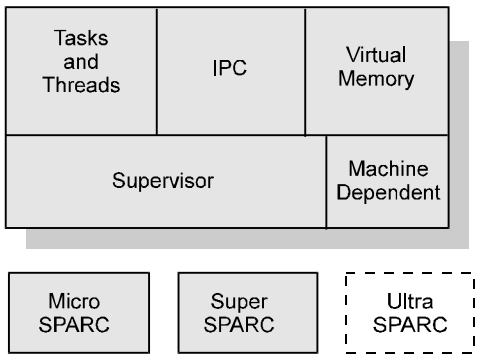


Figure 3.6: PARAS Microkernel

The microkernel integrates the following essential services:

- An executive - derived from the Mach microkernel developed at Carnegie Mellon University - which preemptively schedules priority-based threads.
- A location transparent interprocess communication mechanism.
- A simple virtual memory model.
- A low-level hardware supervisor which dispatches interrupts and traps.
- Standard message passing interfaces like PVM, MPI have been efficiently layered over the microkernel. CORE, a custom messag passing interface is also available over the microkernel.

3.3.2 Servers

The power of PARAM 9000 in MPP mode is provided to the user with the help of two sets of servers; one running on the Service Partition (SP) and the other running on the Compute Partition (CP) (Figure 3.7). The operating environment includes the microkernel running on compute nodes and Solaris 2.x running on service nodes.

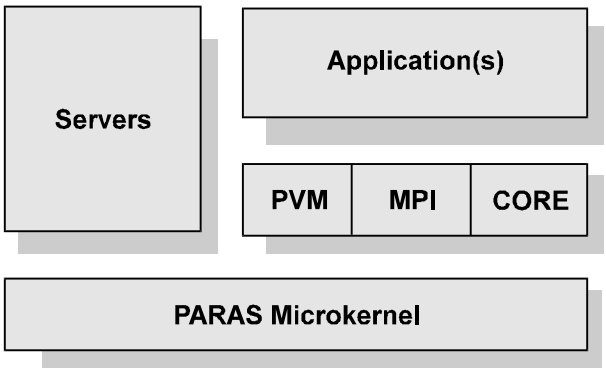


Figure 3.7: Microkernel and Servers

Service Partition Servers

The PARAM 9000 is a multi-user, networked parallel supercomputer with a vast amount of computing and I/O resource at its disposal. A software environment on PARAM 9000 allows the user and system administrator to use and monitor the system effectively. Since the compute partition runs the microkernel, a front-end server running on the service partition allows downloading of user tasks onto the compute partition and service its I/O requests. Param Resource Manager (PRM) is the front-end server which does the above job.

The term PARAM Resource Manager (PRM) is broadly applied to refer to a collection of servers and client tools/utilities that manage, monitor and avail the resources of the PARAM. The PRM runs on the service partition and is in-charge of allocating resources on user requests, provide system administration capabilities (including adding, deleting and retiring user accounts), offer Accounting and Quota Management services. It takes care of loading user tasks and service I/O requests, queue user jobs for batch processing. A system monitoring tool is also provided to deliver on-line information about the system such as system load, partition usage, the list of current users and the status of their jobs etc. The system monitoring tool also provides off-line services like system configuration, user accounts and quota management, and batch processing configuration and management, etc.

The following are the different servers supported by PRM:

Resource Server (RS)
Database Server (DBS)
Pload Server (PLOAD)
Batch Server (BS)
Quota Server (QS)
Accounting server (ACS)
Monitoring server (MS)

Resource Server: It is responsible for resource allocations on the system and works in cooperation with PM (which exists on compute partition) to allocate resources.

Database Server: It is responsible to perform the following operations:

- Maintaining databases recording
- Accounting information
- Batch processing jobs
- Resource allocation
- Partition usage

PLOAD Server: It is front-end tool with which user interacts. That is, it provides user interface to the PARAM user. When user submits applications, it downloads the same to the nodes for execution in coordination with processor server. It also services the I/O requests of the executing application in coordination with the MKFS.

Batch Server: It is responsible for queuing and scheduling of user applications for later execution.

Quota Server: It keeps track of quotas against each user, quotas per partition, and quotas per node.
Accounting Server: It maintains user accounting information.

Monitoring Server: It is responsible for providing on-line information about the following:

- System load
- List of users who have logged onto the system
- Partition usage
- Resource allocated

Compute Partition Servers

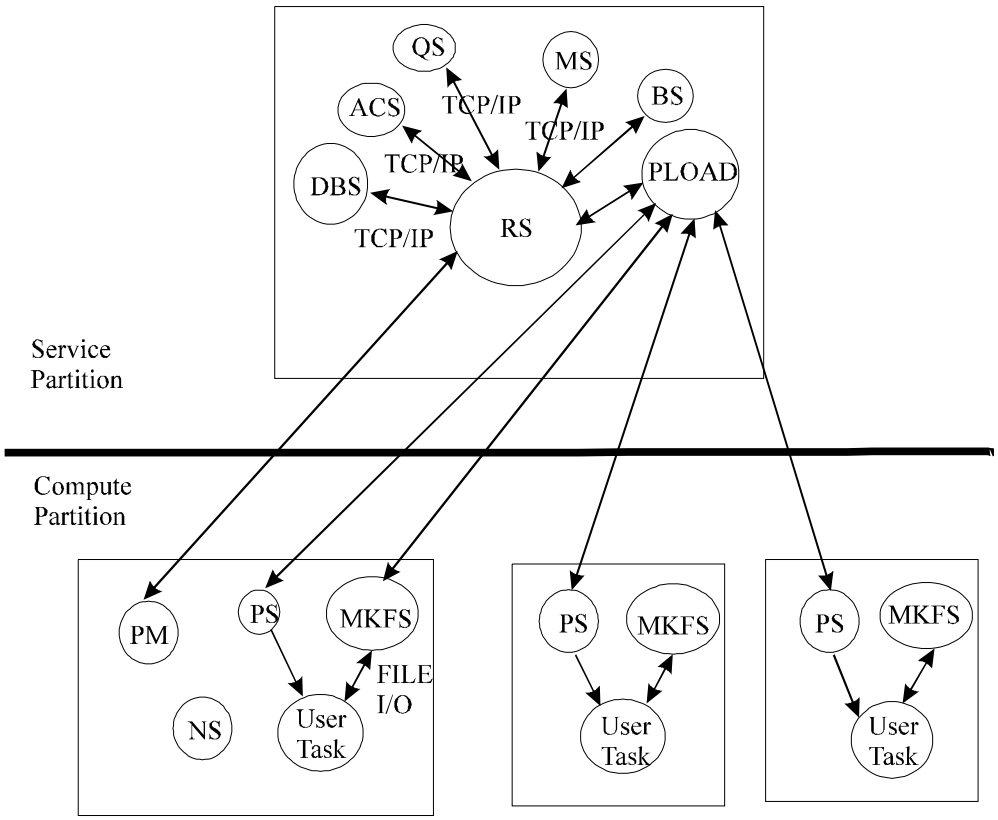
The collection of servers running on the compute partition include the Partition Manager(PM), Process Server (PS), Name Server (NS) and MicroKernel Filesystem Server (MKFS).

Partition Manager serves as a global resource manager for a partition and is responsible for management of that partition's resources. It maintains information about the user tasks, system servers, memory usage on each of the nodes in the Partition. In addition to this it also maintains the system configuration information. PM runs on any one of the compute nodes of a partition.

Each compute node has a file system server and a process server running on it. The filesystem server services the Solaris file system requests on behalf of the user tasks on the compute nodes. The process server provides the required functionality for the user to spawn tasks on a node and also to operate upon them. It also provides a remote system call interface using which user can access the microkernel facilities, from the service partition itself. In addition to these system servers other servers specific to the programming model under execution are also loaded.

Job Execution

The servers on service and compute partitions and their interaction is shown in Figure 3.8. The main servers under the PRM are Resource Server (RS) and Database Server (DBS). All PRM servers run on every node in the SP except the DBS which runs on a designated node. The primary interface of the user is through the set of client utilities which interacts with PRM servers. Pload is a client utility which user executes to run his application programs on PARAM 9000. The user submits the list of tasks and the associated configuration information to Pload, which then contacts the RS for resource allocation. RS in turn contacts Partition Manager (PM) after validating the request. The PM allocates requisite number of nodes which is returned to Pload via RS. Pload then contacts the corresponding Process Server (PS) and downloads the application tasks. Pload then waits to service all I/O requests from the executing tasks until all of them exit.



Figiure 3.8: System Servers in MPP mode

3.3.3 Communication Subsystems

PVM and MPI message passing libraries callable from FORTRAN and C are supported on PARAM 9000 MPP (Figure 3.9). Both the protocols are layered over the microkernel with support from necessary PARAS servers to launch PVM and MPI applications. Any of the PVM or MPI processes can access

files on service nodes through a UNIX like filesystem interface. In addition to PVM and MPI, CORE a custom message interface is also supported.

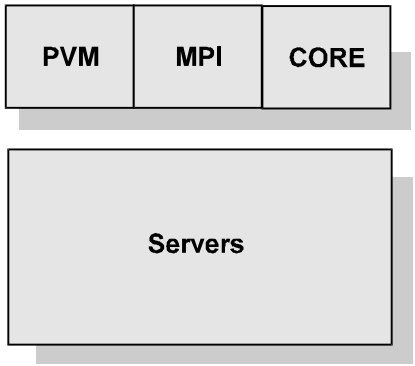


Figure 3.9: Communication Subsystem (MPP)

COncurrent Runtime Environment (CORE)

CORE is an efficient software layer which provides a simple and powerful interface to all system services. User programs can make appropriate system calls to get the necessary services. Through CORE, software developed on earlier versions of PARAM, migrate seamlessly to PARAM 9000. CORE includes support for thread management, semaphore operations, interprocess communication, asynchronous/synchronous I/O and other facilities to access system resources. All these facilities are provided in the form of functions callable from C or Fortran programs.

The message communication takes place via ports, a queue of yet-to-be delivered messages. In addition to both synchronous and asynchronous modes of communications, blocking and non-blocking modes of communication are also supported. Other communication function supported are multicast, barrier synchronization and global operations. CORE also provides a UNIX like interface to file system.

CORE is modular, distributed and highly optimized to reduce the system overheads to a minimum. It makes the programs easy to modify, portable and in many cases more efficient.

3.4 Microkernel Emulator

On PARAM, application tasks are distributed across service and compute partitions; application consists of a collection of cooperative tasks. It is necessary to have a uniform and consistent model for interprocess communication both on a service and compute partition. On compute partition, IPC model is supported by the PARAS microkernel itself whereas, on service partition, IPC model is made available by a *microkernel emulator* (MKE) with a set of user level library functions.

The MKE is a streams upper multiplexer. It supports both the point-to-point (and group) communication using ports. The following are the system calls emulated by the microkernel emulator:

- ◆□□□□□□□□□□□□□
- ◆□□□□□□□□□□□□□
- ◆□□□□□□□□□□□
- ◆□□□□□□□□□□□□
- ◆□□□□□□□□□□□□□□□

These calls are supported by a user-level library which translates them by using standard UNIX system calls such as getmsg(), putmsg(), etc. into appropriate streams messages. These messages in turn send downstream into the MKE. The MKE interacts with the CCP (C-DAC Communication Processor) driver on the lower side which passes messages over the communication network.

3.5 Utilities

The following are the client-side utilities available on the PARAM 9000 supercomputer for the management of tasks:

Application Loader (□□□□□)
Process Kill (□□□□□)
Process Status (□□□)
Process Suspend (□□□□□□□□)
Process Resume (□□□□□□□)

3.5.1 Process Loader: pload - Load the tasks on compute partition

Syntax:

□□
□□

Description:

The command □□□□□ loads the tasks on Compute Partition nodes (Microkernel nodes) in the MPP mode. To run this command, the PRM servers must be running. The user tasks are spawned on the PARAM 9000 compute nodes running PARAS microkernel (CP) nodes through the command “pload” from the host machine (solaris side, SP).

Options:

The following types of options can be submitted to pload from the command line:

□□□□□□□□□□□□□□□□□□□□: The pload loads the tasks on the partition specified in partition name.

□□□□□□□□□□□□□□□: The pload loads the tasks on the specified node type. For example, node type may be SPARC or ALPHA, etc.

□□: Debug option.

Remarks:

The task can be submitted for execution either by directly specifying at the command line or through the configuration file. The pload can be invoked in the following two ways:

(i) By submitting task list at the command line:

Ex: □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

(ii) By submitting tasks through configuration file:

Ex: □□□□□□□□□□□□□

The configuration file must have □□□□ as an extension.

The option □□ is used to debug the tasks using C-DAC’s AIDE (Advanced Integrated Debugging Environment). AIDE uses X-windows as a Graphical User Interface. The pload loads debug kernel and debug servers along with the other user tasks. This option may be used only to debug the tasks on Compute Partition.

Note the environment variable [illegible] should be set when -g option is supplied to the [illegible].

Configuration File Format:

The configuration file allows the user to define the following:

- Task file names
- Mode of execution (Exclusive or Shared)
- Stack size and Heap size

If the mode is defined as EXCLUSIVE, then that particular task (only one task) will run on that specified node and hence, no other tasks are allowed to run on that particular node.

The values of stack size and heap size are to be given in terms of kilo bytes (KB). The user can load any task on any of the nodes. The maximum number of tasks that can be loaded on each node should not exceed 6. The configuration file can have comments embedded into it and they follow the commenting style of C language.

In the configuration file user can specify arguments for the task followed by the task name.

The [illegible] will take the environment variable from the current shell.

If the -g option is specified with pload command, all the tasks of that pload can be debugged.

Example configuration file:

[illegible]

[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]

3.5.2 Process Kill: pkill - kill the tasks running on MPP mode

Syntax:

[illegible]
[illegible]

Description:

The command [illegible] kills the tasks running on MPP mode. It runs only when PRM servers are running. It sends a signal to the specified task running on the Compute Partition side and kills that tasks.

To kill the tasks that are owned by other users, the user need to have supervisor permission. If the -u option is specified, pkill deletes the tasks owned by that particular user.

Options:

The following options can be submitted to the pkill from the command line:

[illegible]: kill the tasks owned by the users in the userlist.
[illegible]: kill the tasks in the tasklist.
[illegible]: kill the tasks spawned by the pload in the ploadlist.
[illegible]: kill the tasks that are loaded on the partition in the Partitionlist.

3.5.3 PARAM Process Status: pps - display the status of tasks running on MPP mode.

Syntax:

[illegible]

Description:

The command [illegible] displays information about tasks running in MPP mode. It runs only when PRM servers are running. Normally, only those tasks that are running with your effective user ID and are attached to a controlling terminal are shown. Additional categories of tasks can be added to the display using various options. In particular, the -a option allows you to include tasks that are not owned by you. The -r option restricts the list of tasks printed to running tasks on MPP mode.

The pps displays the taskID, under TID; ploadID, under PLOADID; the state of the process under STAT; and finally, the name of the task under TASKNAME.

The state is given by a following letters :

R	Running tasks
S	Sleeping tasks
r	Tasks with ready state
I	Tasks that are idle
W	Tasks waiting for I/O
E	Exited tasks

T	Terminated tasks
U	Abnormally terminated tasks
A	Active tasks

Options:

The following options can be submitted to the □□□ from the command line:
-a: Include information about tasks owned by others.
-e: Display the arguments as well as the environment variables
-l: Display more information about the tasks. It displays the partition name, parent process ID, etc.
-r: Restrict the display to the running tasks.
-u <userlist>: Displays the tasks owned by the users in the userlist.
-t<tasklist>: Displays the status of the tasks in the tasklist.
-p <ploadlist>: Displays the tasks spawned by the pload in the ploadlist.
-P <Partitionlist>: Displays the tasks that are loaded on the partition in the Partitionlist.

Note: The state of task can change while pps is running; the picture it gives is only a close approximation to the current state.

3.5.4 Process Suspend: psuspend - suspend the tasks running on MPP mode.

Syntax:

□□□□□□□□□□□□□□□□□

□□

Description:

The command □□□□□□□□ suspends the tasks running on MPP mode. It runs only when PRM servers are running. The □□□□□□□□ sends a signal to the specified task running on the compute partition and suspends that tasks.

To suspend the tasks that are owned by other users, supervisors permission is needed. If the -u option is specified, psuspend suspend the tasks owned by that particular user.

Options:

The following options can be submitted to the psuspend from the command line:
-u <userlist>: suspend the tasks owned by the users in the userlist.
-t <tasklist>: suspend the tasks in the tasklist.
-p <ploadlist>: suspend the tasks spawned by the pload in the ploadlist.
-P <Partitionlist>: suspend the tasks that are loaded on the partition in the Partitionlist.

3.5.5 Process Resume: presume - resume the tasks running on MPP mode.

Syntax:

□□□□□□□□□□□□□□□□

□□

Description:

The command □□□□□□□ resumes the tasks running on MPP mode. It runs only when PRM servers are running. It sends a signal to the specified task running on the compute partition and resumes that tasks.

To resume the tasks that are owned by other users need to be supervisor permission. If the -u option is specified, presume resume the tasks owned by that particular user.

Options:

The following options can be submitted to the [illegible] from the command line:

[illegible]: resume the tasks owned by the users in the userlist.

[illegible]: resume the tasks in the tasklist.

[illegible]: resume the tasks spawned by the pload in the ploadlist.

[illegible]: resume the tasks that are loaded on the partition in the Partitionlist.

3.8 PRM Library Functions

The library functions supported by PRM are discussed below:

[illegible]: Open connections with PRM database and with Micro-kernel servers.

[illegible]: Get task information from the PRM database.

[illegible], [illegible], [illegible]: Kill/suspend/resume the tasks running on the CP.

[illegible]: Close connections with PRM database and with Micro-kernel servers.

Syntax:

[illegible]

Header File: [illegible]

Description:

The [illegible] and prmclose() have to be invoked before invoking any of the prmlib calls. It initializes connections with PRM servers. The [illegible] is used to query the process information Database. The structure [illegible] contains the following fields:

[illegible]

and [illegible] contains the following fields:

[illegible]

□□
□□
□□
□□

The fields pkey and skey of the □□□□□□□□□□□□ structure are to be filled with a constant values specified in the header file ptask.h and the values of primary and secondary key are to be filled in □□□□□□□□□□□ structure. The □□□□□□□□□□□ returns as many number of records that matches the primary and secondary key values; gettinfo returns number of records fetched from the database. If the flag value is BLOCK, gettinfo call will wait for constant time to get the record only if the record is not found at the time of the call is made and the flag value is NOBLOCK then □□□□□□□□□□□ returns immediately without waiting for that matching record, if the record is not found.

The □□□□□□□□ kills the task on the PARAS microkernel with the task_id tid and node_id nid.

The □□□□□□□□□□□ and presume are used to suspend and resume respectively the task on the PARAS microkernel with the specified task_id tid and the node_id nid.

Returns:

On success, these functions except □□□□□□□□□□□ returns 0. On failure, it returns -1 and sets □□□□□□□ to indicate the error. On success gettinfo returns the number of matching records found in the database.

Errors:

These functions fail if any of the following are true:

□□□□□□□□: Unable to connect with DataBase server.
□□□□□□□□□□: Peer Connection Closed.
□□□□□□□□: Improper key value or Key value not found for searching record in the PRM database.
□□□□□□□□□□: No such record in the PRM database.
□□□□□□□□□□: Insufficient memory or Improper pointer value passed to these functions.
□□□□□□□□: No such task in the PRM database.
□□□□□□□□: The effective user of the calling process does not match the real user and is not super-user.
PMKE*: MKE call failure. for more details see □□□□□□□□□□□

Note that before calling these functions, □□□□□□□□□□ and □□□□□□□□□□ functions should be called.