CS6400 Domains and Partitions

Omar Hassaine, Cray Business Systems, San Diego, CA

ABSTRACT: *The domains feature on the Cray Superserver 6400 provides the ability to subdivide the machine into separate single-board systems, which can later be reconfigured back into the rest of the machine. The partitions feature provides the capability to partition the CS6400 processing power into disjoint processor sets that can be independently scheduled. Partitioning also provides support for establishment of minimum working sets of memory. This paper covers the basic design issues, the user's expectations, and a high level description of the user interface provided for both features.*

1 Introduction

The *domains* and *partitions* features of the Cray Superserver 6400 (CS6400) help the system administrator deliver the power of 64 SPARC processors to meet his user's needs. Domains split the machine into separate systems each running a distinct instance of Solaris Unix. Partitions allocate the processors within one system to different portions of that system's workload.

1.1 Domains

Domains are a well-known feature in the mainframe world. They provide the ability to run a separate instance of the operating system on a portion of the hardware that has been isolated from the rest of the system. Domains on the CS6400 are created by isolating a single system board from the rest of the system. This system board (with four processors, a gigabyte of memory, and four SBus I/O slots) is then a completely independent Solaris system.

Domains add to the availability of the CS6400 by creating a separate environment for safely bringing up and testing new software. A domain may be used to test and debug new mission-critical application software, or to break-in a new Solaris release without impacting production users. Because a domain is physically isolated from the four-XDBus system interconnect, software and hardware errors inside the domain cannot affect the rest of the system. Equally, errors in the remaining portion of the system will not affect the domain.

Because each domain is its own computer system, it must have appropriate peripheral and network connections. Each domain must be configured with a disk to boot from, a network connection, and sufficient memory and disk space to accomplish the intended task.

A system board can be dynamically detached from the main system — without affecting that systems's continuing operation — to create a new domain.

Solaris can then be booted up on the new domain, and it is available for immediate use. Each domain has its own host-id, and operates as a completely separate computer system. Single-board domains can be created, used separately, and then later added back into to the main system — without requiring a re-boot of the main system:

1. The operator requests that a system board be logically detached. The kernel flushes all the processes, swaps out the user pages, and physically remaps the kernel pages away from the board. The board's XDBus interface is set to ignore bus transactions.
2. The operator boots a new instance of the kernel on the single board. The board is used as a separate system, for example, to use a new release of Solaris.
3. Later, when the single-board system is no longer needed, the operator shuts down that instance of Solaris.
4. The operator requests that the board's XDBus interface begin again to send and receive XDBus transactions to and from other system boards. The multi-board kernel begins to use the inserted board by adding that board's processor, memory, and I/O resources back into the system.

1.2 Partitions

Partitions allow flexibility in workload management. They allow a system administrator to split up the disjoint parts of a workload between groups of processors so that resources are predictably divided among users. Historically this capability was provided in the processor scheduling algorithms with the goal of guaranteeing a certain percentage of each processor to specific workloads. In a cached system with many processors, such as the CS6400, it is much more efficient to dedicate a

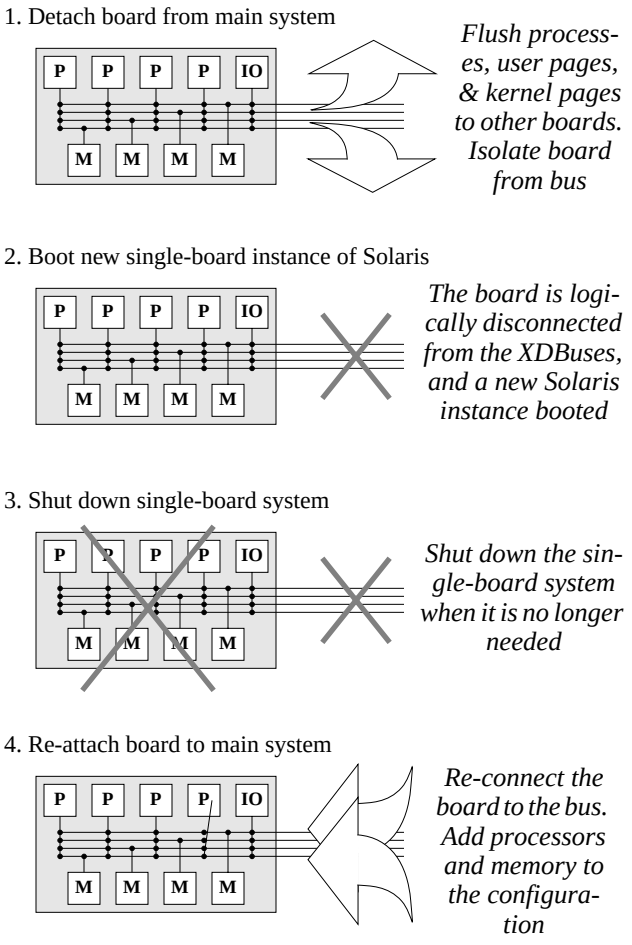

Figure 1: Creating and removing domains

percentage of the processors to a workload than to dedicate a percentage of each processor to a workload. Space sharing is more efficient than timesharing.

The system administrator can set up the partitions, giving them access permissions and symbolic names, and assigning processors and attributes to them. He then assigns particular process sets to run in particular partitions, or allows users to choose their own partitions from among those for which they have permission.

The processors in a partition give priority to the processes assigned to that partition. Optionally, idle processors can temporarily borrow processes from other partitions, and can loan processes to idle partitions. A partition can be set up to service SBus I/O interrupts, or it can just compute.

The system administrator can execute the partitioning commands dynamically, or can use scripts to partition the CS6400 system during the boot process. She may choose to assign specific processes, such as the NFS server and client daemons, to run in a specific partition. Users may assign themselves to specific partitions as they log in, or may be automatically assigned to partitions based on their user ID or upon partition loading..

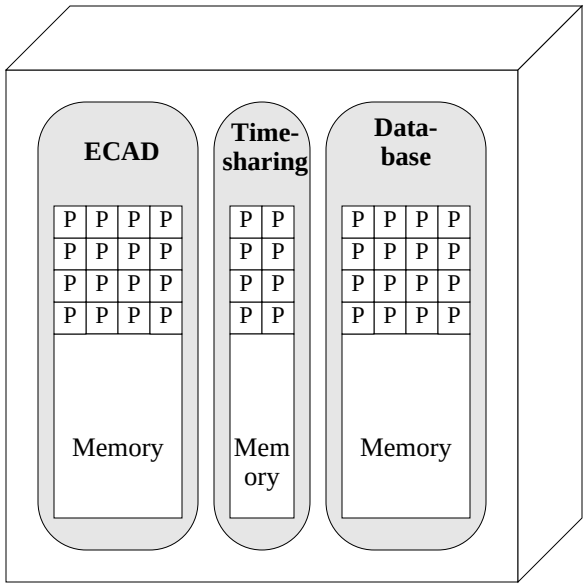

Figure 2: Example of processor partitioning

Partitioning can also help debug parallel applications programs that tend to hang the system by restricting them to run in a dedicated partition while debugging from another partition.

While we were in the design phase for our next-generation product at Cray Business Systems, we chose to divide our CS6400 server into two eight-processor partitions: one for integrated circuit design and simulation, and the other for general timesharing. This split kept either set of users from hogging the machine's resources.

2 Design issues

The two features are distinct from an implementation and user interface point of view. Domains was implemented as part of the System Service Processor software and its related commands execute strictly on the System Service Processor side. Partitions was implemented in the Solaris kernel and its related commands and programs execute only on the CS6400.

2.1 Domains design issues

The domains project was a challenge for the design team because the original system software requirements did not include the domains concept. The major requirements that were laid out for the domains project were:

1. There will be one master domain and up to 15 slave domains.
2. Only the master domain can have more than one board and is not created using the domains user interface.
3. The slave domains execute only a subset of the System Service Processor commands and daemons.
4. No change to the user interface.

5. Domains can run different Solaris versions.
6. Dynamic Reconfiguration is a required feature for domains.

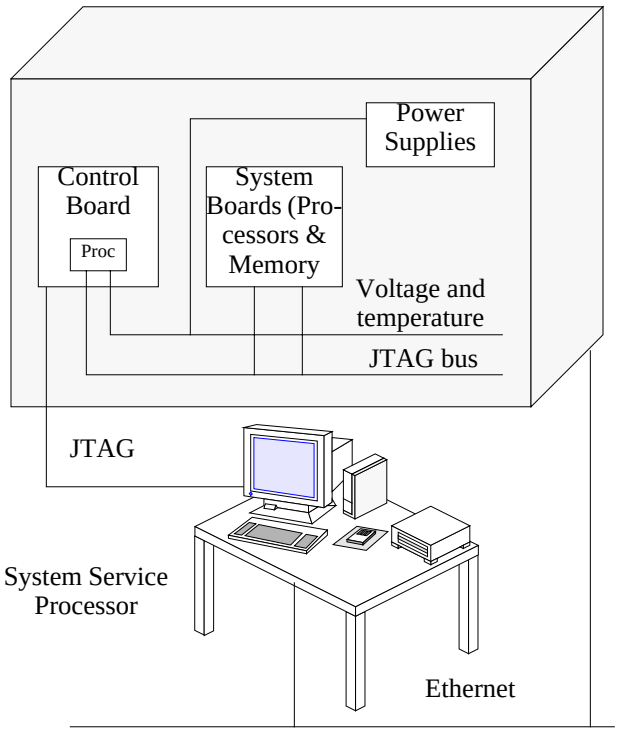

Figure 3: CS6400 with System Service Processor

The following gives a list of the major issues that we resolved in order to successfully complete the domains project:

2.1.1 Hostmon

Hostmon is the main daemon that runs on the System Service Processor and is in charge of collecting various information about the CS6400 and taking actions accordingly. Hostmon had to be modified to support more than one machine within the CS6400. We worked on the event dispatcher and make it check which domain the event will affect.

2.1.2 Obp_helper

Obp_helper is a daemon that cooperates with the boot firmware (OBP) and performs the task of the network console server. This daemon had to be modified to run as a separate copy for each created domain. We wrote another daemon (the machine_server) that makes sure that console sessions talk to the corresponding obp_helper.

2.1.3 Cray Virtual Console (CVC)

CVC is a network based console user session and requires a TCP/IP port for communication with the CS6400 through the *obp_helper* daemon. The modifications here required that the console session passes the CRAY_HOSTNAME environment variable to the *machine_server* to get assigned the next available TCP/IP port from the pool that is dedicated for all possible domains that are allowed to be created in a CS6400.

2.1.4 Support for multiple-versions of Solaris

The most challenging requirement for domains was for the System Service Processor software to be able to monitor multiple domains that run different versions of Solaris. We came up with a scheme that divided the System Service Processor commands into domain-independent and domain-dependent categories, as illustrated in Figure 4.

The domain-independent set can be executed in the System Service Processor environment that controls the master domain. The domain-dependent set of commands can be executed in the System Service Processor environment that controls the domain whose host name is set to the `CRAY_HOSTNAME` environment variable.

The Domain-dependent commands include a peculiar class of System Service Processor commands called the Solaris-version-dependent commands. These commands have an intimate relationship with the Solaris operating system running on the CS6400. For each version of Solaris supported, there may be a separate version of the Solaris-version-dependent System Service Processor commands residing in the software package. The software packaging on the System Service Processor had to be redesigned to fit the new version scheme so that the execution of Solaris-version-dependent commands is transparent to the user. We put a constraint on all the non-Solaris-version-dependent commands to be backward compatible so that only a single binary is needed regardless of the different version of Solaris running in the various domains.

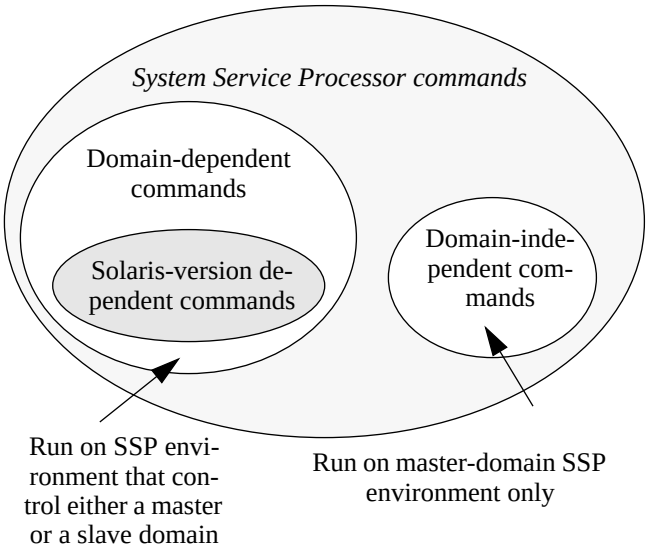

Figure 4: System Service Processor command categories

2.1.5 Hostview

The Graphical User Interface was significantly augmented to support domains. A new paradigm had to be devised to allow the various System Service Processor tasks to take place if and only if a domain is selected from the main hostview screen. The pre-domains hostview had no concept of select first and then perform operation.

2.1.6 *Dynamic reconfiguration*

Dynamic reconfiguration [3] (DR) is a feature that allows a board to be dynamically attached and detached from a running system. The domains feature introduced a new machine configuration file (`machine_config`) that reflects the various domain configurations running on the host. The format of each file entry is as follows:

```
domain_name:Os_version:system_board_list
```

The dynamic-reconfiguration operation needs to update the `machine_config` file after each successful operation. In the attach case, the board number of the attached board needs to be added to the `system_board_list` of the master domain. In the detach case, the board number must be removed from the corresponding list. The correct behavior of the domains feature relies on the integrity of the `machine_config` file.

2.1.7 *Boot heuristics*

Boot heuristics is an algorithm used by Hostmon to reboot the machine by degrading its physical configuration following multiple panics in a relatively short period. With the introduction of the domains feature, the boot heuristic algorithm was modified to affect only the master domain and make sure that a reboot action will not try to include a live slave domain board into a newly reconfigured master domain.

2.2 Partitions design issues

The design of partitions focused mostly on how to minimize extensions to the various parts of the kernel by adhering to the following requirements:

1. The number of partitions is at most equal to the number of processors in the system.
2. The boot processor (*cpu0*) always belongs to the system partition and system threads always run in this partition.
3. Workload sharing can occur among partitions that allow it.
4. The Solaris scheduling strategy is preserved within each partition except for thread borrowing/loaning between consenting partitions.
5. Threads inherit partition assignment from their parent.
6. Provision for user and application interface to the various partitioning services.
7. Each partition can be assigned its own memory Resident Set Size (RSS).

The implementation of this feature has impacted several areas in the kernel and modified major kernel data structures. The rest of this section gives a brief overview of the main modifications required to support the partitioning feature:

2.2.1 *Dispatcher*

The Solaris thread dispatcher assigns threads to the various processors for execution. The algorithm used by the original dispatcher had to be modified to take note of which partition a thread is assigned and restrict it to run only in its partition or in another partition with the proper attribute.

2.2.2 *p_online(2)*

The `p_online()` system call changes the on-line or off-line status of a processor and can query the status of a processor. When a processor is put off-line it does not lose its partition assignment. This system call had to be modified to support the case when the very last processor of a partition is put off-line. Typically, when a processor is put off-line, its processes are reassigned to the remaining processors in the same partition. If the last processor in a partition is put off-line then we must flush out all of the processes running in the partition and assign them to the system partition.

2.2.3 *Dynamic reconfiguration*

Dynamic reconfiguration affects also the partitioning feature. When a board is attached, its processors are automatically added to the system partition. During the detach operation, the processors need to be put off-line one at a time by following the rules laid out in the previous section.

2.2.4 *Processor/process related functions*

All the processor/process related system calls such as `processor_bind`, `processor_info` and others are affected by this feature. Their return values are evaluated in a context of partition. A system call invoked by a non-privileged user will cover only the set of processors/processes belonging to the partition. A reference to a processor outside of the non-privileged partition will be treated as a reference to an off-line processor. A process with effective superuser uid can reference all processors in the system. A privileged process running in one partition, for instance can bind a Light Weight Process (LWP) to a processor in another partition.

2.2.5 *Processor_info(2)*

The `processor_info()` system call returns the type and status of a processor. A non-privileged call to `processor_info()` has access to the processors in the partition of the calling light-weight process. Other processors in the system appear as off-line processors.

A privileged call to `processor_info()` has access to all processors in the system.

2.2.6 *Changes to the cpu structure*

`cpu_partition_mask`. A bit mask of the processors in the partition to which the processor belongs

`cpu_partitionp`. A pointer to the partition structure for the processor (a two-byte index into the partition table)

2.2.7 *Changes to the thread structure*

Two fields were added to the thread structure:

1. `t_partition_mask`. A bit mask of the processors in the partition to which the thread is assigned.
2. `t_partitionp`. A pointer to the partition structure for the partition to which the thread is assigned. (a two-byte index into the partition table).

2.2.8 Changes to disp_cpulist handling

The function of the `disp_cpulist` array is to provide a dispatching processor a quick way to determine which processors have generally runable high priority threads enqueued. If the highest priority is higher than any thread enqueued to the dispatching processor, the dispatching processor steals it.

2.2.9 New data structures

The partition table is an array of 64 partition structures. A partition structure has the following fields:

- partition ID
- partition lock
- processor bit mask
- partition attributes
- pointer to the `disp_cpulist` for this partition
- `disp_cpulist_lock` variable for this partition
- pointer to the `disp_cpu_map` variable for this partition
- `max_unbound_pri` for this partition

3 User Interface

3.1 Domains user interface

The domains user interface can be executed either from the command line interface or the Hostview Graphical User Inter face (GUI). The GUI is a much simpler interface to use and it only takes a couple of steps; the first one is to select the board used as a domain and any function selected will apply to that domain only. As to the command line interface, Figure 5: illustrates the user interface:

You can configure any CS6400 machine for domains provided the following conditions are met:

- The board is present and not in use.
- The board has a network interface
- The board has sufficient memory to support itself as an autonomous host
- The name given to the new domain is unique.

3.1.1 Domain creation

Many of the instructions from the `suninstall` section of the Solaris installation guide have been modified to reflect Cray specific changes. The System Service Processor User's Guide describes step by step the creation and removal of domains. This section will be limited to basically cover the main logical steps that lead to the creation of a domain. It is also assumed that the administrative logistics on the host side is ready for booting the single board domain.

1. Logically disconnect the system board from the host by using the Dynamic Reconfiguration detach or gracefully shut down the host, blacklist the board and then bringup the CS6400.
2. If the board is blacklisted, unblacklist it and execute the `domain_create` command:

   domain_create domain_name board_number

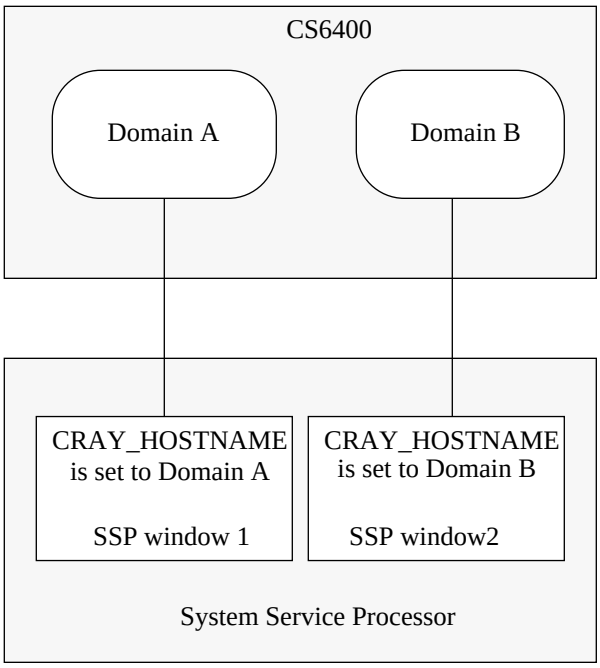

Figure 5: SSP user interface windows with domains

3. Create a System Service Processor window where the environment variable `CRAY_HOSTNAME` is set to *domain_name*. In this window, all commands executed such as: `bringup`, `sysreset`, `check_host` will all be applied only to the domain created.

3.1.2 Domain Removal

This operation can be performed either from Hostview or from the command line interface. Removing a domain basically consists of first gracefully shutting down the domain and run the domain_remove command on the System Service Processor environment that controls the master domain.

3.1.3 Domain Status

It is possible to view the status of the system with respect to domains from the Hostview GUI. Each domain board is highlighted with a different color and can also be viewed independently from the menu bar `view` function. No command line user interface was implemented to show the status of domains. However; the contents of the `domain_config` file hold the information on each domain present in a particular system.

3.2 Partitions

In this section, we will present an overview on partition management, Light Weight Processes assignment to partitions and miscellaneous partition functions. This section is particularly useful to a system administrator that wants to get familiar with the partitioning feature. The manual page for `partitioning(1M)` and `partn(1M)` in the CS6400 reference related

to partitions are the next step because they cover the mechanics of partitions administration.

3.2.1 Partition Management

This section will discuss a generic case of system partitioning that will illustrate the concept of partitions management. The example below covers all the specific attributes related to processor and physical memory assignment that are provided by the partitioning feature.

Figure 6 shows the example of 64 processors that have been partitioned into four sets of 16 processors each (P1, P2, P3, P4) and each of the partitions has been assigned a memory Resident Set Size (RSS). Thread set A always runs on processors in P0 unless a processor in the other partitions, P1 and P2, has no work. Since partition P0 can loan threads to other partitions and partitions P1 and P2 can borrow, processors in P1 and P2 can borrow threads from P0 when they are idle. Since P0 does not have the attribute *can borrow*, it is restricted to running threads assigned to partition P0 only. Since partitions P1 and P2 do not loan, other partitions cannot run threads assigned to P1 or P2. Thus thread set B is confined to run in partition P1 and thread set C is confined to run in partition P3.

System Partition P0 Processors 0-15 Memory RSS: 2 GB	*Partition P1* Processors 16-31 Memory RSS:1 GB
Partition P2 Processors 32-47 Memory RSS: 1 GB	*Partition P3* Processors 48-63 Memory RSS: 2 GB

Thread Assignments

Thread Set A: Partition P0 with attributes *can loan*

Thread Set B: Partition P1 with attributes *can borrow*

Thread Set C: Partition P2 with attributes *can borrow*

Thread Set D: Partition P3

Figure 6: Example partitions

Partition P3 is a dedicated partition. It neither borrows nor loans. Processors belonging to P3 can only execute thread set D, and thread set D can run in partition P3 only.

3.2.2 Assigning LWPs and processes to partitions

By default all threads in the system are assigned to the system partition. Light-weight processes may be assigned to a different partition in the following groupings:

1. current LWP
2. the indicated LWP in the current process
3. all LWPs of the current process
4. all LWPs of the indicated process
5. all processes of the indicated uid
6. all processes of the indicated process group
7. all processes of the indicated session.

A light-weight processes can be assigned to a single partition only. An assignment of a light-weight processes to a partition replaces its previous partition assignment.

A bound light-weight processes cannot be assigned to a partition that does not contain the processor to which the light-weight processes is bound — it must first be unbound.)

By default a processor having bound light-weight processes cannot be reassigned to a different partition. This rule can be explicitly overridden.

3.2.3 Miscellaneous functions

Partition information. The information that can be retrieved from the system about partitions is the number of partitions, the partition ID of each partition, its processor set bit mask, and its attributes.

Altering partitions. Processors can be added to a partition. This operation causes the indicated processors to be removed from their current partitions and placed in the specified partition. If this operation causes a partition to become empty, the threads assigned to the partition are assigned to the system partition.

Clearing partitions. When a partition is cleared its members are returned to the system partition. The system partition may not be emptied.

4 Programmatic Interface for partitions

The domains feature does not support an application interface. However; the partitioning feature does provide a header file and an exhaustive list of library calls that allow applications dynamic access to partitioning services.

4.1 Partitions library header file

The file `/usr/include/partn_mgt.h` is provided and contains the definition of the interface exported by the library routines. The function prototypes, manifest constants and structure definitions are in this file

4.2 Partition Management Library Routines

Partition management library routines are provided to assign processors to partitions, to set the attributes of partitions and to display information about partitions. Note that *read* access to a partition is required to display information about it and *root* privilege is required for the other partition management operations.

4.2.1 partn_open

Open the partition and return the partition number. The partition number is the minor device number of the partition's pseudo device. A partition must be opened prior to accessing it.
The partition number is used as the first parameter to the partition management routines.

Name may be `[0-63]` or a symbolic link to `/dev/partn/[0-63]`. The `partn_open` routine looks in the directory /dev/partn for the filename.

4.2.2 partn_close

Closes the partition with minor device number indicated by the partition number. It is semantically similar to closing a file.

4.2.3 partn_get_number

Given a partition filename return the partition number. The symbolic links in `/dev/partn` to the partitions `[0-63]` provide the translation mechanism from symbolic name to partition number.

4.2.4 partn_get_name

Given a partition number return the partition name. Places the leaf filename of the symbolic link to the `/dev/partn/[0-63]` file into a buffer *buf* of size *bufsiz*. The value of *bufsize* must be at least one greater than the filename to be returned. The symbolic link must be in the same directory as the `/dev/partn/[0-63]` files. If there is no symbolic link, the leaf filename of the `/dev/partn/[0-63]` file is returned.

4.2.5 partn_assign_processors

Assigns the processors referenced in the `cpulist` array to a partition. If the force flag is not set, this operation will fail if there are any light-weight processes bound to one of the processors. If the force flag is set, the processor will be moved and the light-weight processes bound to it will be reassigned to run in the new partition.

4.2.6 partn_clear

Removes all processors from the partition. Moves the processors in partition *pno* to the system partition. Light-weight processes assigned to the partition are reassigned to run in the system partition. If the force flag is not set, this operation will fail if there are any light-weight processes bound to a processor in the partition. If the force flag is set, the processor will be moved and the light-weight processes bound to it will be reassigned to run in the new partition.

4.2.7 partn_get_processors

Places the processor IDs of the processors assigned to a partition into a buffer and returns the number of processor IDs returned.

Note that the upper bound of *ncpus* can be determined by calling `sysconf(3C)` naming `_SC_NPROCESSORS_CONF`.

4.2.8 partn_set_attributes

Set the attributes of a partition. The three possible attributes are as follows:

1. `PARTN_ATTR_CAN_BORROW`. If a partition becomes idle, it is allowed to temporarily borrow work from other partitions that allow it.
2. `RTN_ATTR_CAN_LOAN`. The partition allows idle processors in other partitions to temporarily borrow light-weight processes to run.
3. `PARTN_ATTR_NO_SBUS_INTS`. The processors in the partition will not participate in SBus interrupt servicing. This attribute is invalid for the system partition.

4.2.9 partn_get_attributes

Get the attributes of a partition. Same as for `partn_set_attributes()` plus `PARTN_ATTR_SYSTEM`: `PARTN_ATTR_SYSTEM`. This partition is the system partition. Only partition 0 has this attribute. It cannot be assigned to a different partition.

4.2.10 partn_snap_runqlen

Get a snapshot of the length of the partition's dispatch queue. It returns the current length of the partition's dispatch queue. This is the sum of the lengths of the dispatch queues of all the processors in the partition.

4.2.11 get_ref_processor

Get the processor ID of the reference, or boot processor. The reference processor is the processor that the system booted on. It cannot be removed from the system partition.

4.3 Process Management Functions

This section describes functions that assign light-weight processes and processes to partitions and retrieve information about light-weight processes and processes. Assigning processes to run in partitions requires write permission to the partition and the ability to signal the process. Retrieving information requires read access to the partition.

4.3.1 process_assign_to_partn

Assigns the light-weight processes or processes specified to run in a particular partition. The set of light-weight processes or processes to which this function applies is specified by *idtype* (`type idtype_t`) and an *id* list (a list of *ids* of `type id_t`). These types are defined in `<sys/procset.h>`.

Note that system processes (pids 0, 2, and 3) may not be moved from the system partition.

The caller must have write access to the partition and the ability to signal the process.

4.3.2 procset_assign_to_partn

This routine is exactly like `processor_assign_to_partn` except for the manner in which the set of light-weight processes and processes to assign to a partition are specified by a structure of type `procset_t` (refer to `priocntl(2)`).

4.3.3 process_get_partn

Get the number of the partition to which an light-weight processes or a process is assigned. The scope of each of this function is a single light-weight processes or process, which is indicated by the *idtype* and *id* parameters. The *idtype* parameter can be either `P_LWPID` or `P_PID`.

If *idtype* is `P_PID` the function applies to the process with process ID `id`.

If *idtype* is `P_LWPID` the function applies to the light-weight processes of the current process with light-weight processes ID `id`.

If *id* is `P_MYID` the specified process or light-weight processes is the current one.

4.3.4 process_snap_lwpcnts

For a given light-weight processes or process, get a snapshot of the number of its light-weight processes and bound light-weight processes running in the indicated partition.

4.4 Memory Management Functions

This section describes functions that manage memory resident set sizes.

4.4.1 partn_get_prss

Upon successful completion, `partn_get_prss` returns the partition's reserved resident set size. The special value of `0x0` denotes an unlimited reserved resident set size and is the default value when a new partition is created.

4.4.2 partn_set_prss

Set a partition's resident set size reservation. The size of the reserved resident set size is in megabytes. The special value of `0x0` is used to denote an unlimited resident set size as described earlier.

4.4.3 partn_get_crss

This routine returns the current resident set size of a particular partition. This is the sum of the resident set sizes of all processes in the partition.

5 Future enhancements

5.1 Domains

The domains feature was added after the System Service Processor software was released, so that several trade-offs were made. The list below attempts to give an overview of the various enhancements planned for the future:

1. The concept of master (main) domain and slave domains will be eliminated so all domains are handled equally from the System Service Processor software side. A consequence of this concept is that there will be no restriction on certain System Service Processor commands to be executed from only a particular domain environment.
2. Provision for domain status and domain history commands. These commands will help the administrator have an easy interface to find out the status of domains in a particular physical box. The history of inactive domains can help save eeprom information about a certain domain and avoid confusion of host ID information for later reuse.
3. Improve the System Service Processor software to make it as independent as possible from the Solaris version in order to control domains that run different versions of operating systems.
4. Provide support for more than a three domain configuration in the same machine. Due to the System Service Processor architecture that relates to polling JTAG, a performance penalty was incurred when more than three domains are configured. A new System Service Processor software architecture is being designed that will be event driven and distributed will allow more domains to be configured.

5.2 Partitions

This feature was implemented by surgically coding the Solaris kernel and keeping the modifications to the possible minimum. Due to schedule pressures, the feature was not enhanced to a more desirable level. Possible improvements would be:

1. Write a dynamic load balancer daemon that periodically checks the workload in the various partitions and uses some heuristic algorithm to reassign processes across consenting partitions.
2. The concept of the system partition that contained the boot processor made it easy to implement the feature but this has hampered the assignment of system threads to other partitions.
3. Solve the idle partition problem by periodically checking for inactive partitions and reallocating some of its processors to other partitions that desperately need the power.
4. Implement a set of scheduling policies (fair share, gang scheduling, dynamic load balancing, etc.) for different partitions. The selection of a scheduling policy together with a partition will depend upon the intended use of the partition.

6 Summary and Conclusion

This paper gave an overview of the issues encountered during the design phase of the related domains and partitioning enhancements. Domains divide the machine into multiple systems that act as separate machines with their own operating system, hostname, IP address, boot disk and so on. Partitioning divides the processing power of one system into disjoint groups that may or may not share workloads and memory.

The software implementation of domains was strictly done on the System Service Processor side with the help of the Dynamic Reconfiguration feature, whereas the partitioning feature was strictly done at the kernel level on the host side.

The paper also gave a high level description of the user interface provided by both features and the application interface supported by the partitioning feature.

7 References

[1] *CS6400 Domains Design Specifications*, Cray Research Inc. Internal Documentation, 1994.

[2] *Process Control Extensions for CS6400: Functional Specifications*, Cray Research, Inc. Internal Documentation, 1994.

[3] *Process Control Extensions for CS6400: Design Overview*, Cray Research Inc. Internal Documentation, 1994

[4] *CS6400 System Service Processor User's Guide*. Cray Research Inc, 1994.