

Processor and Memory API Report

January 25, 2007

Copyright © 2004 Hal's Software Inc.

Contents

1	Introduction	4
1.1	Basic Overview	4
1.2	Program Optimizations	4
1.3	Informational Interfaces	5
1.4	Control Interfaces	5
1.4.1	Processor Binding APIs	5
1.4.2	Processor-Set APIs	6
1.4.3	Memory Allocation and Placement APIs	6
2	System Support	7
2.1	Microsoft Windows	7
2.1.1	Basic System Information	7
2.1.2	NUMA System Information	7
2.1.3	Advisory Processor Association	8
2.1.4	Processor Binding	8
2.2	Sun Solaris	8
2.2.1	Basic System Information	8
2.2.2	NUMA System Information	8
2.2.3	Advisory Processor Association	8
2.2.4	System Control	8
2.2.5	Processor Binding	9
2.2.6	Global Processor Sets	9
2.2.7	Page Size Manipulation	9
2.2.8	Memory Migration and Placement	9
2.3	HP-UX	9
2.3.1	Basic System Information	9
2.3.2	NUMA System Information	9
2.3.3	Advisory Processor Association	10
2.3.4	Processor Binding	10
2.3.5	Global Processor Sets	10
2.3.6	Memory Migration and Placement	10
2.4	IBM AIX	10
2.4.1	Basic System Information	10
2.4.2	NUMA System Information	10
2.4.3	Configuration Change Notification	11
2.4.4	Processor Binding	11
2.4.5	Global Processor Sets	11
2.4.6	Page Size Manipulation	11
2.4.7	Memory Migration and Placement	11
2.5	SGI IRIX	11
2.5.1	Basic System Information	11
2.5.2	NUMA System Information	11
2.5.3	System Control	12
2.5.4	Advisory Processor Association	12
2.5.5	Processor Binding	12
2.5.6	Global Processor Sets	12
2.5.7	Page Size Manipulation	12
2.5.8	Memory Allocation and Placement	12
2.6	Tru64 UNIX	13
2.6.1	Basic System Information	13
2.6.2	NUMA System Information	13
2.6.3	Advisory Processor Association	13
2.6.4	Processor Binding	13
2.6.5	Global Processor Sets	14

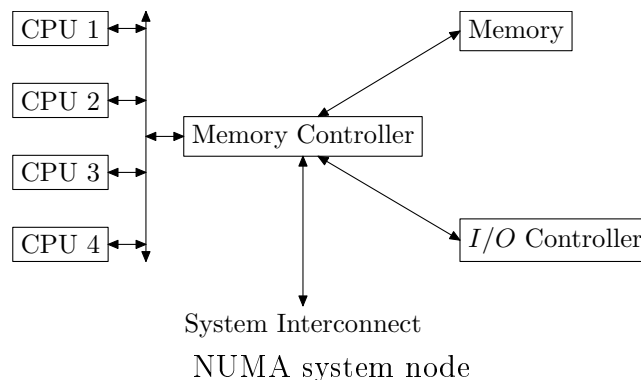
2.6.6	Memory Migration and Placement	14
2.7	Linux	14
2.7.1	Basic System Information	14
2.7.2	NUMA System Information	14
2.7.2.1	SGI cpumemsets	14
2.7.3	Processor Binding	14
2.7.3.1	LinuxThreads	15
2.7.3.2	NPTL	15
2.7.3.3	SGI cpumemsets	15
2.7.3.4	RTLinux	15
2.7.4	Memory Migration and Placement	15
2.7.4.1	SGI cpumemsets	15
2.8	QNX	15
2.8.1	Basic System Information	15
2.8.2	Processor Binding	15
2.9	BSD Derivatives	16
2.9.1	Basic System Information	16
3	Acknowledgments	17
	References	18
	Index	21

Chapter 1

Introduction

1.1 Basic Overview

Traditionally, systems with multiple processors have been designed using the symmetric multiprocessor (SMP) model. In this model, no individual processor or group of processors have faster access to system memory or I/O devices than any other processor in the system. Unfortunately, the scalability of an SMP setup is limited by the finite bandwidth of the singular processor bus used to connect the processors to each other and the other devices on the system. To overcome the scalability problems associated with SMP designs, systems with a large number of processors often use a design which features non-uniform memory access (NUMA). In a NUMA system, CPUs are grouped into smaller subsystems called nodes. Each node may contain a small number of processor, memory and I/O devices and is connected to other nodes in the system using an interconnect bus. As a result of the physical system topology, each processor and I/O device has faster access to memory located on the same or in nearby nodes and slower access to memory located in nodes which are farther away. Most NUMA systems use cache-coherent NUMA (ccNUMA) designs which insure that each processor has globally consistent view of system memory.



Operating systems and other software vendors have developed various APIs to allow user-level programs to examine system topology, control thread scheduling and optimize memory allocation. Although these different APIs often share many similarities, they are not governed by industry standards and differ from each other in significant ways. Additionally, each vendor provides administrative control programs which differ in both interface and functionality from similar programs by other vendors.

1.2 Program Optimizations

When a variety of different applications with differing resource needs are run on the same system, it is often best to rely on the default behavior of the operating system to reactively schedule threads on different processors and allocate memory for those threads appropriately. However, many applications

may realize performance benefits through the preemptive use of APIs which control thread scheduling and memory allocation.

Applications with many processes or threads which operate on different subsets of a large dataset may benefit from the use of APIs which control thread scheduling and memory allocation on NUMA systems. Thread scheduling APIs can be used to force threads to run on processors with fast access times to the dataset(s) which the threads are accessing frequently. Likewise, the application can change the memory allocation policies of the operating system to help insure that relevant partitions of the dataset(s) are placed in memory near the processors which are executing the threads from which data will be most frequently accessed.

In general, thread scheduling APIs should be used to insure that threads which are heavily dependent on shared data run on the same node. Additionally, shared data should be partitioned and/or replicated whenever possible such that the data is available locally to the threads running on each node which must access the data frequently. Applications which execute large I/O requests may benefit from the use of APIs which control memory allocation on NUMA systems supporting multipath I/O (MPIO). Multipath I/O refers to a setup in which a physical device such as a disk is attached to more than one adapter in the system, possibly in different nodes, thus creating multiple access paths to the device. Unless specific information regarding the device topology is available, memory used in large I/O requests should be distributed across all available system nodes to maximize performance. When an application accesses a portion of its address space, the processor must translate the requested virtual memory address into a physical memory address. This translation process is expensive, but the processor can cache the results on a page-by-page basis. In general, the cache can only hold information on a fixed number of translations. When an application frequently accesses more memory pages than can be held in the translation cache, its performance may be significantly degraded. Such applications will likely benefit from the use of APIs which control memory page size. By increasing the size of memory pages used to manage specific portions of the application's address space it is possible to significantly increase the application's performance.

1.3 Informational Interfaces

Almost all vendors provide APIs which at least allow a program to determine the number of processors available to the program on the system. Some also allow programs to access additional system topology information. On such systems, optimizations which take advantage of the hierarchical organization of the system can prove surprisingly fruitful. Although operating systems generally do a reasonable job at scheduling processes and allocating memory to take advantage of system configuration, there are many instances where better resource utilization can be achieved by asserting control over scheduling and memory allocation using program specific knowledge.

1.4 Control Interfaces

Available control interfaces can be broken down into several categories: processor-binding APIs, processor-set APIs, and APIs to control memory allocation and placement.

1.4.1 Processor Binding APIs

Processor binding APIs associate a collection of processors with each thread and/or process and only allow the thread or process to be scheduled to run on processors in the set. If the system allows the set to consist of more than one processor then it is usually represented by a bit-mask of processors. On some systems, the binding can be made advisory instead of mandatory.

1.4.2 Processor-Set APIs

Processor sets are global collections of processors to which a thread and/or process can be assigned. Once assigned, the thread or process will only be scheduled to run on processors within the set. Exclusive processor sets allow an individual processor to belong to only one set at a time.

1.4.3 Memory Allocation and Placement APIs

The capabilities provided for memory allocation control vary widely. Some systems provide the capability to control the set of nodes from which memory is allocated. Also, some systems allow threads to request that parts of their address space be placed on specific nodes.

Chapter 2

System Support

The following sections discuss the relevant interfaces provided by specific systems.

System	Basic System Information	NUMA System Information	Configuration Change Notification	Advisory Processor Association	Processor Binding	Global Processor Sets	Page Size Manipulation	Memory Migration and Placement
Windows	✓	✓	—	✓	✓	—	—	—
Solaris	✓	✓	—	✓	✓	✓	✓	✓
HP-UX	✓	✓	—	✓	✓	✓	—	✓
AIX	✓	✓	✓	—	✓	✓	—	✓
IRIX	✓	✓	—	✓	✓	—	✓	✓
Tru64 UNIX	✓	✓	—	✓	✓	✓	✓	✓
Linux	✓	✓	—	—	✓	✓ ¹	—	✓ ¹
QNX	✓	—	—	—	✓	—	—	—
FreeBSD ²	✓	—	—	—	—	—	—	—

¹ With cpumemsets or similar extension.

² Also OpenBSD, NetBSD and MacOS X (Darwin).

2.1 Microsoft Windows

2.1.1 Basic System Information

The `GetSystemInfo` function can be used to retrieve the number of processors on the system, the list of active processors and the memory page size.

2.1.2 NUMA System Information

Windows Server 2003 introduced NUMA support. System topology information is provided by the `GetLogicalProcessorInformation`, `GetNumaHighestNodeNumber`, `GetNumaNodeProcessorMask`, `GetNumaProcessorNode` and `GetNumaAvailableMemoryNode` functions. The current processor

can be determined using the `GetCurrentProcessorNumber` function. The provided topology information can also include information about SMT (Hyperthreaded) logical processors.

2.1.3 Advisory Processor Association

Starting with Windows NT 4.0, support was introduced for a thread to have a preferred processor using the `SetThreadIdealProcessor` function. A thread's current ideal processor is stored in the thread's `KTHREAD` structure ([[NTIFS](#)]).

2.1.4 Processor Binding

All systems starting with Windows NT 3.5 (and Windows 95) support thread scope binding using the `SetThreadAffinityMask` function. Additionally, these systems support process scope binding in the form of the `GetProcessAffinityMask` function. The characteristics of these functions differ slightly on the Windows 95 family of operating systems as these systems do not include multi-processor support.

Windows NT 4.0 and higher systems support provide for the manipulation of the process affinity mask using the `SetProcessAffinityMask` function. The current thread affinity mask can be queried on Windows 2000, Windows XP and Windows Server 2003 systems using the undocumented thread information class `ThreadBasicInformation` with the `NtQueryInformationThread` function to get a copy of the thread's `THREAD_BASIC_INFORMATION` structure ([[NTDLL](#)]).

2.2 Sun Solaris

2.2.1 Basic System Information

The `sysconf` function can be used with the `_SC_CPUID_MAX`, `_SC_NPROCESSORS_CONF`, `_SC_NPROCESSORS_MAX` flags to determine basic system processor configuration. The current state of a processor can be determined using the `processor_info` function. The `getpagesize` function returns the default memory page size. A list of available memory page sizes can be retrieved using the `getpagesizes` function.

2.2.2 NUMA System Information

Newer versions of Solaris Operating Environment 9 (starting December 2002) support the Memory Placement Optimization (MPO) API. This API allows a thread to determine its current processor using the `getcpuid` function and its current home latency group (current node) using the `gethomegroup` function. Additionally, newer versions of the Solaris Operating Environment support the Locality Group API. This API is exported through the `lgrp` library. NUMA topology information can be retrieved using the `lgrp_view`, `lgrp_children`, `lgrp_parents`, `lgrp_root`, `lgrp_nlgrps`, `lgrp_cpus` and `lgrp_mem_size` functions.

2.2.3 Advisory Processor Association

The `lgrp_affinity_get` and `lgrp_affinity_set` functions can be used to get and set a thread's local group association. Each process and thread is assigned a home group used for default scheduling purposes. The home group can be retrieved using the `lgrp_home` function.

2.2.4 System Control

The Solaris Operating Environment allows online processor status changes using the `p_online` function. This function allows a processor to be switched between online and offline states. Solaris supports the concept of a non-interruptible online processor state in which the processor is schedulable but will not be used to process external I/O events.

2.2.5 Processor Binding

The Solaris Operating Environment supports the concept of process and thread processor binding. The function `processor_bind` can be used to set or query a process's or thread's processor binding.

2.2.6 Global Processor Sets

The Solaris Operating Environment also supports exclusive processor sets. These sets are created and manipulated using the `pset_create`, `pset_destroy`, `pset_assign` and `pset_setattr` functions. Information regarding a given processor set can be retrieved using the `pset_info` and `pset_getattr` functions. The processor set binding can be set or queried using the `pset_bind` function. System load information specific to a given processor set can be retrieved using the `pset_getloadavg` function. A list of processor sets can be retrieved using the `pset_list` function.

2.2.7 Page Size Manipulation

Before Solaris 9, allocating memory using a large page size could be done using the `SHM_SHARE_MMU` flag with the `shmat` function. This feature is known as intimate shared memory (ISM). On Solaris 9 and latter systems, specific page sizes can be requested through the use of the `MC_HAT_ADVISE` flag with the `memcntl` function.

2.2.8 Memory Migration and Placement

Information regarding the associated latency group of memory regions can be obtained using the `meminfo` function with the `MEMINFO_VLGRP` or `MEMINFO_VREPL_LGRP` flags. Memory region placement can also be optimized using the `madvise` function with the `MADV_ACCESS_LWP`, `MADV_ACCESS_MANY` or `MADV_ACCESS_DEFAULT` flags. Specifically, using the `MADV_ACCESS_LWP` flag, a given thread can claim a region of virtual memory for placement to optimize access from the thread's current latency group.

2.3 HP-UX

2.3.1 Basic System Information

The number of processors on the system can be determined using the `MPC_GETNUMSPUS_SYS` flag with the `mpctl` function. Each processor on the system has a unique ID. The IDs of all system processors can be enumerated using the `MPC_GETFIRSTSPU_SYS` and `MPC_GETNEXTSPU_SYS` flags with the `mpctl` function. The number of processors available to a particular thread can be determined using the `pthread_num_processors_np` function or using the `MPC_GETNUMSPUS` flag with the `mpctl` function. The IDs of processors available to a particular thread can be determined using the `PTHREAD_GETFIRSTSPU_NP` and `PTHREAD_GETNEXTSPU_NP` flags with the `pthread_processor_bind_np` function or using the `MPC_GETFIRSTSPU` and `MPC_GETNEXTSPU` flags with the `mpctl` function.

2.3.2 NUMA System Information

The `_SC_CCNUMA_SUPPORT` flag can be used with the `sysconf` function to determine if the system has NUMA support. The `MPC_GETNUMLDOMS_SYS`, `MPC_GETFIRSTLDMO_SYS`, `MPC_GETNEXTLDMO_SYS`, `MPC_LDOMSPUS` and `MPC_SPUTOLDOM` flags can be used with the `mpctl` function to retrieve system topology information. A process can determine its current processor using the `MPC_GETCURRENTSPU` flag with the `mpctl` function. The number of NUMA nodes available to a particular thread can be determined using the `pthread_num_ldoms_np` function or using the `MPC_GETNUMLDOMS` flag with the `mpctl` function. The ID of each node available to a particular thread can be determined using that `PTHREAD_GETFIRSTLDMO_NP` and `PTHREAD_GETNEXTLDMO_NP` flags with the `pthread_ldom_id_np` function or using the `MPC_GETFIRSTLDMO`

and `MPC_GETNEXTLDDOM` flags with the `mpctl` function. The number of available processors within each such node can be determined using the `pthread_num_ldomprocs_np` function or using the `MPC_LDDOMSPUS` flag with the `mpctl` function.

2.3.3 Advisory Processor Association

The `MPC_SETPROCESS` flag can be used with the `mpctl` to associate a process with a particular processor. The `PTHREAD_BIND_ADVISORY_NP` flag can be used to associate a thread with a particular processor.

2.3.4 Processor Binding

A process can be bound to a specific processor using the `MPC_SETPROCESS_FORCE` flag with the `mpctl` function. Similarly, a process can be bound to a specific NUMA node using the `MPC_SETLDDOM` flag with the `mpctl` function. A thread can be bound to a specific processor using the `PTHREAD_BIND_FORCED_NP` flag with the `pthread_processor_bind_np` function. A thread can be bound to a specific NUMA node using the `pthread_ldom_bind_np` function.

2.3.5 Global Processor Sets

Support for exclusive processor sets and processor binding have been included since HP-UX 11i version 1.6. Processor sets are created and manipulated using the `pset_create`, `pset_destroy`, `pset_assign` and `pset_setattr` functions. Information regarding a given processor set can be retrieved using the `pset_ctl` and `pset_getattr` functions. The `pset_ctl` function can also be used to obtain a list of current system processor sets and processor topology information. Support for processor sets can be queried by using the `_SC_PSET_SUPPORT` flag with the `sysconf` function. The `pset_bind` function is used to bind a particular process to a processor set. A thread can be bound to a processor set using the `pthread_pset_bind_np` function.

2.3.6 Memory Migration and Placement

The `mmap` and `shmget` functions have been enhanced to accept the `(MAP|IPC)_MEM_INTERLEAVED`, `(MAP|IPC)_MEM_LOCAL` and `(MAP|IPC)_MEM_FIRST_TOUCH` flags.

2.4 IBM AIX

2.4.1 Basic System Information

The number of system processors or online system processors can be determined using the `sysconf` function with the `_SC_NPROCESSORS_CONF` or `_SC_NPROCESSORS_ONLN` flags respectively. Alternatively, the variables `_system_configuration.ncpus` or `_system_configuration.max_ncpus` can be used. The default memory page size can be determined by using the `_SC_PAGESIZE` flag with the `sysconf` function. The `_SC_LARGE_PAGESIZE` flag can be used with the `sysconf` function to determine the size of large memory pages.

2.4.2 NUMA System Information

The number of NUMA nodes available to a particular program can be determined using the `rs_numrads` function. Information about a specific node can be retrieved using the `rs_getinfo` function. Detailed topology information can be obtained through the use of the `rs_getassociativity` function.

2.4.3 Configuration Change Notification

Processes which are bound to a processor will be notified by the `SIGRECONFIG` signal if the state of the processor is scheduled to change. Within the signal handler, the application should call the `dr_reconfig` function to obtain the details of the impending change. Applications with appropriate credentials can cancel the change before it goes into effect.

2.4.4 Processor Binding

Applications and/or threads can bind to a processor using the `bindprocessor` function. Kernel thread IDs can be retrieved using the `thread_self` function. Additionally, the mapping between kernel thread ids and pthread handles can be accessed using the functions `pthdb_pthread_tid` and `pthdb_tid_pthread`. Threads bound to a CPU which is being deallocated will be sent the `SIGCPUFAIL` signal. The `ra_attachrset` function or the `rs_setpartition` function can be used to bind a process to a specified NUMA node. The `ra_exec` and `ra_fork` functions allow processes to be created with such bindings.

2.4.5 Global Processor Sets

AIX allows for arbitrary groupings of processors and memory units known as resource sets. The `rs_op` function is used in combination with the `rs_init`, `rs_alloc` and `rs_getrad` functions to create and manipulate arbitrary resource sets. Global resource sets are assigned names with the `rs_setnameattr` function and retrieved using the `rs_getnamedrset` function.

2.4.6 Page Size Manipulation

An application can allocate memory using the large page size by using the `SHMLGPGPAGE` and `SHMPIN` flags with the `shmget` function. Large page support must be enabled by the system administrator using the `vmtune` command.

2.4.7 Memory Migration and Placement

As AIX resource sets can include memory units, it is possible to use resource sets to restrict the set of NUMA nodes from which application memory is allocated. Also, it is possible to force migration of application memory from one set of nodes to another set using resource-set bindings.

2.5 SGI IRIX

2.5.1 Basic System Information

Basic configuration information can be retrieved using the `MP_NPROCS`, `MP_NAPROCS` and `MP_STAT` flags with the `sysmp` function. The current default memory-management policy set can be accessed using the `pm_getdefault` function.

2.5.2 NUMA System Information

IRIX provides a namespace for processors and nodes under the `/hw` directory. Nodes are identified by, for example, `/hw/module/1/slot/n1/node`, with aliases as, for example, `/hw/nodenum/0 -> /hw/module/1/slot/n1/node`. Processors are identified by, for example, `/hw/module/1/slot/n1/node/cpu/a` where cpus are named either `a` or `b` within their nodes. Processor aliases are available as, for example, `/hw/cpunum/0 -> /hw/module/1/slot/n1/node/cpu/a`. The hardware graph

filesystem is not guaranteed to be mounted under `/hw`, so it may be necessary to search for the mount point of `hwgfs` using the `getmntent` function.

2.5.3 System Control

The `sysmp` function allows a processor's state to be changed to exclude it from running processes not specifically bound to it. It is also possible to assign the processor used to manage the system clock. The `MP_ISOLATE` flag allows a processor to delay cache synchronization until system services are requested. The `MP_NONPREEMPTIVE` and `MP_WARDRTC` flags allow a processor not to process clock and other timer events.

2.5.4 Advisory Processor Association

When a memory locality domain is linked to a process using the `process_mldlink` with the `RQMODE_ADVISORY` flag, then the binding is considered advisory.

2.5.5 Processor Binding

Using the `sysmp` function with the `MP_MUSTRUN` or `MP_MUSTRUN_PID` flags, it is possible to bind a process to an isolated processor. The function `pthread_setrunon_np` allows binding of a thread with `PTHREAD_SCOPE_SYSTEM` or `PTHREAD_SCOPE_BOUND_NP` scope to a particular isolated processor. Binding to NUMA nodes can be accomplished using memory locality domains (MLDs). When a process is attached to a MLD using the `process_cpulink` function, the scheduler attempts to schedule the process on a processor where the MLD has been placed. Memory locality domain binding with `process_cpulink` is not available for `pthread` enabled applications.

2.5.6 Global Processor Sets

IRIX allows processors to be excluded from the running of unbound processes. These processors will only run processes which are explicitly bound to them. Thus, each excluded processor can be thought of as being a single-element exclusive set processor set.

2.5.7 Page Size Manipulation

IRIX supports the concept of a memory-management policy set. The memory page size is only element which can be included in the policy.

2.5.8 Memory Allocation and Placement

IRIX supports the concept of memory locality domains. The function `mld_create` is used to create a memory locality domain of a given size. The function `numa_arena_create` will create a memory arena for memory allocation within the memory locality domain. The memory locality domains are grouped into memory locality domain sets. A set is created using the `mldset_create` function. The function `mldset_place` must be used to place a memory locality domain set using a given topology and resource affinity set. It is possible to request specific topological arrangements close to a given device or file. The function `migr_range_migrate` along with `migr_policy_args_init` can be used to migrate a memory range into a given memory locality domain. Different memory management strategies can be associated with different regions in a program's address space. Policy information can include a memory locality domain set used preferentially for placement and allocation. A policy set is created using the `pm_create` function and attached to a given memory region using the `pm_attach` function. A policy set can be designated the default policy set using the `pm_setdefault` function. The current policy set for a given memory region is accessed using the `pm_getall` function. Policy set parameters can be extracted from a policy set handle using the `pm_getstat` function. The `_pm_get_page_info`

and `_mld_to_node` functions can be used to determine placement information for a given memory region.

2.6 Tru64 UNIX

2.6.1 Basic System Information

The number of system processors and the number of active system processors can be determined using the `sysconf` function with the `_SC_NPROCESSORS_CONF` or `_SC_NPROCESSORS_ONLN` flag respectively. The default memory page size can be determined using the `_SC_PAGESIZE` flag with the `sysconf` function. System processor and basic topology information is available through the `getsysinfo` function.

2.6.2 NUMA System Information

Starting with Tru64 UNIX version 5.1, the system supports the concept of Resource Affinity Domains (RADs) and CPU sets. CPU set utility functions are `cpuaddset`, `cpuandset`, `cpucopysset`, `cpucountset`, `cpudelset`, `cpudiffset`, `cpuemptyset`, `cpufillset`, `cpuisemptyset`, `cpuisemember`, `cpuorset`, `cpusetcreate`, `cpusetdestroy` and `cpuxorset`. The CPUs in a CPU set can be enumerated using the `cpu_foreach` function. The current CPU can be retrieved using the `cpu_get_current` function. System CPU information can be retrieved using the `cpu_get_info`, `cpu_get_num` and `cpu_get_max` functions. The function `cpu_get_rad` can be used to get the RAD associated with a given CPU. Topology information is available through the `nloc` function. The CPU set of a given RAD can be determined using the `rad_get_cpus` function. RAD memory information can be retrieved using the `rad_get_freemem` and `rad_get_phymem` functions. The online/offline state of a RAD can be queried using the `rad_get_state` function. Basic system RAD information can be retrieved using the `rad_get_num` and `rad_get_max` functions. RAD set utility functions are `radaddset`, `radandset`, `radcopysset`, `radcountset`, `raddelset`, `raddiffset`, `rademptyset`, `radfillset`, `radisemptyset`, `radisemember`, `radorset`, `radsetcreate`, `radsetdestroy` and `radxorset`.

2.6.3 Advisory Processor Association

A process can be associated with a given RAD using the `rad_attach_pid` function. A thread can be associated with a given RAD using the `pthread_rad_attach` function.

2.6.4 Processor Binding

Process CPU binding is supported on Tru64 UNIX using the `bind_to_cpu` and `bind_to_cpu_id` functions. A thread can be bound to a specific processor using the `pthread_use_only_cpu` function. A process can be bound to a given RAD using the `rad_bind_pid` function. A thread can be bound to a given RAD using the `pthread_rad_bind` function. The current RAD can be determined using the `rad_get_current_home` function. The function `nfork` can be used to copy the calling process or thread and assign the child process to a different RAD. Also, the `rad_fork` function can be used to fork the current process or thread specifying the RAD of the child process. Tru64 UNIX supports the concept of a NUMA Scheduling Group which allows processes to be bound to specific system nodes. A scheduling group can be created or accessed using the `nsg_init` function. The status of a scheduling group is queried using the `nsg_get` function. Processes are attached to a given scheduling group using the `nsg_attach_pid` function. A list of configured NUMA scheduling groups can be retrieved using the `nsg_get_nsgs` function. A list of processes attached to a given scheduling group can be retrieved using the `nsg_get_pids` function. The owner and permissions of a scheduling group can be set using the `nsg_set` function. Threads can be attached to a scheduling group using the `pthread_nsg_attach` function and detached using the `pthread_nsg_detach` function. A list of threads attached to a given scheduling group can be retrieved using the `pthread_nsg_get` function. Process and RAD binding information for a process can be retrieved using the `numa_query_pid` function.

2.6.5 Global Processor Sets

Processor sets are created using the `create_pset` function. Processors are assigned to processor sets using the `assign_cpu_to_pset` function. Processes are assigned to a given processor set using the `assign_pid_to_pset` function.

2.6.6 Memory Migration and Placement

The memory allocation policy for a given memory region can be queried using the `memalloc_attr` function. The system can be advised as to the expected access pattern of a given memory region using the function `nmadvise`. The `mmap` function allows an open file to be mapped into the process's address space specifying the memory allocation policy. The `nshmgget` function allows a shared memory region to be created or accessed while specifying the memory allocation policy.

2.7 Linux

2.7.1 Basic System Information

Most Linux systems support the retrieval of basic system processor information using the `sysconf` function with the `_SC_NPROCESSORS_CONF` and `_SC_NPROCESSORS_ONLN` flags.

2.7.2 NUMA System Information

Linux 2.5.46 and later systems provide topology information through the `sysfs` file system namespace (`sysfs` was also known as `driverfs` in some earlier releases). Nodes are exported in the `class/node/devices` directory such that, for example, the directory `class/node/devices/0` -> `../../root/sys/node0` has a file named `cpumap` which contains a bitmap of CPUs in a given node. Also `class/cpu/devices` contains symlinks to CPUs, for example, `class/cpu/devices/0` -> `../../../../root/sys/cpu0`. Memory block topology is also represented under the `class/memblk/devices` directory tree such that, for example, `class/memblk/devices/0` -> `../../../../root/sys/memblk0` and `root/sys/node0/memblk0` -> `../../../../memblk0` exist. The `getmntent` function can be used to locate the mount point for `sysfs`.

2.7.2.1 SGI cpumemsets

SGI provides Linux based systems which have been extended to support the `cpumemsets` API. The system uses CPU maps to map system CPU identifiers to process CPU identifiers. The CPU map for a given process can be retrieved using the `cmsQueryCMM` function and set using the `cmsSetCMM` function. The function `cpu2node` is used to determine the node containing a given CPU. Basic system configuration information is provided by the `numnodes`, `numcpus` and `lubcpunum` functions. Basic system configuration information is provided by the `numnodes`, `numcpus` and `lubcpunum` functions. The functions `nodebind`, `cpu2node`, `numnodes`, `numcpus`, and `lubcpunum` depend on the existence of a valid `/var/cpuset/cpu-node-map` file which should be created a boot time by a system initialization script.

2.7.3 Processor Binding

Linux 2.5.8 and later systems support the `sched.getaffinity` and `sched.setaffinity` functions which can be used to get and set the processor affinity of a given process. The processor affinity is represented as a list of processors on which a given process or thread is allowed to run.

2.7.3.1 LinuxThreads

On systems with the LinuxThreads pthread implementation, each thread is given its own PID and thus can be controlled in the same manner as any other system process. The thread debugging interface provides the necessary functions to map a thread handle into a system PID and a system PID into a thread handle using the functions `td_thr_get_info` and `td_ta_map_lwp2thr` respectively (the thread debugging library was designed to emulate the thread debugging library on Solaris). In the LinuxThreads implementation the LWPID of type `lwpid_t` maps to the system PID of type `pid_t` which instantiates the thread.

2.7.3.2 NPTL

On systems using the NPTL (Native POSIX Thread Library) pthread implementation, the function `pthread_setaffinity_np` or `pthread_attr_setaffinity_np` can be used to bind a thread to a set of processors. The function `pthread_getaffinity_np` or `pthread_attr_getaffinity_np` can be used to retrieve the given processor set bindings. The thread debugging library can be used with the NPTL implementation as with the LinuxThreads implementation if necessary.

2.7.3.3 SGI cpumemsets

Each process has a cpumemset which contains a list of processors on which the given process can run and a list of nodes on which a given set of CPUs can allocate memory. Different areas in the process's address space can also be assigned a cpumemset to control memory allocation. CPU maps and cpumemsets returned by the query functions must be freed using the `cmsFreeCMM` and `cmsFreeCMS` functions respectively. The `cmsGetCpu` function returns the current CPU. Several utility binding functions are also provided: `runon`, `cpubind` and `nodebind`.

2.7.3.4 RTLinux

RTLinux, a Linux based system produced by FSMLabs, supports binding threads to a given processor using the `pthread_attr_setcpu_np` function and querying the CPU binding using the `pthread_attr_getcpu_np` function. The current CPU can be determined by using the `rtl_getcpuid` function.

2.7.4 Memory Migration and Placement

2.7.4.1 SGI cpumemsets

Using the SGI cpumemsets API it is possible to assign different cpumemset to different regions of a process's address space. This allows an application to designate different topological policies for different address-space regions.

2.8 QNX

2.8.1 Basic System Information

On a QNX system, information regarding the system topology is accessed using the `_syspage_ptr` global structure instance accessed using the `SYSPAGE_ENTRY(entry)` macro.

2.8.2 Processor Binding

Processor affinity can be set using the `_NTO_TCTL_RUNMASK` flag with the `ThreadCtl` or `ThreadCtl_r` function.

2.9 BSD Derivatives

2.9.1 Basic System Information

BSD Derivatives include FreeBSD, OpenBSD, NetBSD and MacOS X (Darwin). Newer BSD derivative systems support the `_SC_NPROCESSORS_CONF` and `_SC_NPROCESSORS_ONLN` flags to the `sysconf` function. The number of system processors can also be obtained using the `CTL_HW` and `HW_NCPU` flags with the `sysctl` function.

Chapter 3

Acknowledgments

- Windows is a registered trademark of Microsoft Corporation in the United States and other countries.
- Solaris is a trademark or registered trademark of Sun Microsystems, Inc. in the United States and other countries.
- HP-UX is a registered trademark of Hewlett-Packard Company Corporation in the United States and other countries.
- AIX is a registered trademark of International Business Machines Corporation in the United States and other countries.
- IRIX is a registered trademark of Silicon Graphics, Inc. in the United States and other countries.
- Tru64 is a registered trademark of Compaq Computer Corporation in the United States and other countries.
- UNIX is a registered trademark of The Open Group in the United States and other countries.
- Linux is a registered trademark of Linus Torvalds in the United States and other countries.
- RTLinux is a registered trademark of Finite State Machine Labs, Inc.
- QNX is a registered trademark of QNX Software Systems, Ltd.
- BSD is a registered trademark of Berkeley Software Design, Inc.
- FreeBSD is a registered trademark of Wind River Systems, Inc.
- NetBSD is a registered trademark of the NetBSD Foundation.
- MacOS and Darwin are registered trademarks of Apple Computer, Inc.
- All other trademarks and product names are the property of their respective owners.

References

Official Documentation

- [AIXMan] ..
- [AIXManK] ..
- [AIXPage] ..
- [AIXRA] ..
- [HPUXMan] ..
- [IRIXMan] ..
- [MSDN] ..
- [SolMan] ..
- [SolMulPage] ..
- [SolPSet] ..
- [Tru64Man] *Tru64 UNIX Reference Pages*, .

Additional Sources

- [NTDLL] *Undocumented functions of NTDLL* <<http://undocumented.ntinternals.net>>, Tomasz Nowak. **2.1.4**
- [NTIFS] *ntifs.h* <<http://www.insidewindows.info/ntifs.h>>, Bo Brantén. **2.1.3**

Index

- [_NTO_TCTL_RUNMASK](#), 15
- [_SC_CCNUMA_SUPPORT](#), 9
- [_SC_CPUID_MAX](#), 8
- [_SC_LARGE_PAGESIZE](#), 10
- [_SC_NPROCESSORS_CONF](#), 8, 10, 13, 14, 16
- [_SC_NPROCESSORS_MAX](#), 8
- [_SC_NPROCESSORS_ONLN](#), 10, 13, 14, 16
- [_SC_PAGESIZE](#), 10, 13
- [_SC_PSET_SUPPORT](#), 10
- [_mld_to_node](#), 12
- [_pm_get_page_info](#), 12
- [_syspage_ptr](#), 15
- [_system_configuration.max_ncpus](#), 10
- [_system_configuration.ncpus](#), 10

- [assign_cpu_to_pset](#), 14
- [assign_pid_to_pset](#), 14

- [bind_to_cpu](#), 13
- [bind_to_cpu_id](#), 13
- [bindprocessor](#), 11

- [ccNUMA](#), 4
- [cmsFreeCMM](#), 15
- [cmsFreeCMS](#), 15
- [cmsGetCpu](#), 15
- [cmsQueryCMM](#), 14
- [cmsSetCMM](#), 14
- [cpu2node](#), 14
- [cpu_foreach](#), 13
- [cpu_get_current](#), 13
- [cpu_get_info](#), 13
- [cpu_get_max](#), 13
- [cpu_get_num](#), 13
- [cpu_get_rad](#), 13
- [cpuaddset](#), 13
- [cpuandset](#), 13
- [cpubind](#), 15
- [cpucopyset](#), 13
- [cpucountset](#), 13
- [cpudelset](#), 13
- [cpudiffset](#), 13
- [cpuemptyset](#), 13
- [cpufillset](#), 13
- [cpuisemptyset](#), 13
- [cpuisemember](#), 13
- [cpuorset](#), 13
- [cpusetcreate](#), 13
- [cpusetdestroy](#), 13
- [cpuxorset](#), 13
- [create_pset](#), 14
- [CTL_HW](#), 16

- [dr_reconfig](#), 11
- [driverfs](#), 14

- [getcpuid](#), 8

- [GetCurrentProcessorNumber](#), 7
- [gethomegroup](#), 8
- [GetLogicalProcessorInformation](#), 7
- [getmntent](#), 11, 14
- [GetNumaAvailableMemoryNode](#), 7
- [GetNumaHighestNodeNumber](#), 7
- [GetNumaNodeProcessorMask](#), 7
- [GetNumaProcessorNode](#), 7
- [getpagesize](#), 8
- [getpagesizes](#), 8
- [GetProcessAffinityMask](#), 8
- [getsysinfo](#), 13
- [GetSystemInfo](#), 7

- [HW_NCPU](#), 16
- [hwgfs](#), 11

- [IPC_MEM_FIRST_TOUCH](#), 10
- [IPC_MEM_INTERLEAVED](#), 10
- [IPC_MEM_LOCAL](#), 10

- [KTHREAD](#), 8

- [lgrp](#), 8
- [lgrp_affinity_get](#), 8
- [lgrp_affinity_set](#), 8
- [lgrp_children](#), 8
- [lgrp_cpus](#), 8
- [lgrp_home](#), 8
- [lgrp_mem_size](#), 8
- [lgrp_nlgrps](#), 8
- [lgrp_parents](#), 8
- [lgrp_root](#), 8
- [lgrp_view](#), 8
- [lubcpunum](#), 14

- [MADV_ACCESS_DEFAULT](#), 9
- [MADV_ACCESS_LWP](#), 9
- [MADV_ACCESS_MANY](#), 9
- [madvise](#), 9
- [MAP_MEM_FIRST_TOUCH](#), 10
- [MAP_MEM_INTERLEAVED](#), 10
- [MAP_MEM_LOCAL](#), 10
- [MC_HAT_ADVISE](#), 9
- [memalloc_attr](#), 14
- [memcntl](#), 9
- [meminfo](#), 9
- [memory page size](#), 5
- [migr_policy_args_init](#), 12
- [migr_range_migrate](#), 12
- [mld_create](#), 12
- [mldset_create](#), 12
- [mldset_place](#), 12
- [mmap](#), 10
- [MP_ISOLATE](#), 12
- [MP_MUSTRUN](#), 12
- [MP_MUSTRUN_PID](#), 12

- MP_NAPROCS, 11
- MP_NONPREEMPTIVE, 12
- MP_NPROCS, 11
- MP_STAT, 11
- MP_WARDRTC, 12
- MPC_GETCURRENTSPU, 9
- MPC_GETFIRSTLDM, 9
- MPC_GETFIRSTLDM_SYS, 9
- MPC_GETFIRSTSPU, 9
- MPC_GETFIRSTSPU_SYS, 9
- MPC_GETNEXTLDM, 9
- MPC_GETNEXTLDM_SYS, 9
- MPC_GETNEXTSPU, 9
- MPC_GETNEXTSPU_SYS, 9
- MPC_GETNUMLDM, 9
- MPC_GETNUMLDM_SYS, 9
- MPC_GETNUMSPUS, 9
- MPC_GETNUMSPUS_SYS, 9
- MPC_LDMSPUS, 9
- MPC_LDMSPUS_SYS, 9
- MPC_SETLDM, 10
- MPC_SETPROCESS, 10
- MPC_SETPROCESS_FORCE, 10
- MPC_SPUTOLDOM, 9
- mpctl, 9
- MPIO, 5
- multipath I/O, 5

- nfork, 13
- nloc, 13
- nmadvise, 14
- nmmmap, 14
- nodebind, 14, 15
- nsg_attach_pid, 13
- nsg_get, 13
- nsg_get_nsgs, 13
- nsg_get_pids, 13
- nsg_init, 13
- nsg_set, 13
- nshmget, 14
- NtQueryInformationThread, 8
- NUMA, 4
- numa_acreate, 12
- numa_query_pid, 13
- numcpus, 14
- numnodes, 14

- optimization, 4

- p_online, 8
- page size, 5
- pm_attach, 12
- pm_create, 12
- pm_getall, 12
- pm_getdefault, 11
- pm_getstat, 12
- pm_setdefault, 12
- process_cpulink, 12
- process_mldlink, 12
- processor_bind, 9
- processor_info, 8
- pset_assign, 9, 10
- pset_bind, 9, 10
- pset_create, 9, 10
- pset_ctl, 10
- pset_destroy, 9, 10
- pset_getattr, 9, 10
- pset_getloadavg, 9
- pset_info, 9
- pset_list, 9
- pset_setattr, 9, 10
- pthdb_pthread_tid, 11
- pthdb_tid_pthread, 11
- pthread_attr_getcpu_np, 15
- pthread_attr_setaffinity_np, 15
- pthread_attr_setcpu_np, 15
- PTHREAD_BIND_ADVISORY_NP, 10
- PTHREAD_BIND_FORCED_NP, 10
- pthread_getaffinity_np, 15
- PTHREAD_GETFIRSTLDM_NP, 9
- PTHREAD_GETFIRSTSPU_NP, 9
- PTHREAD_GETNEXTLDM_NP, 9
- PTHREAD_GETNEXTSPU_NP, 9
- pthread_ldom_bind_np, 10
- pthread_ldom_id_np, 9
- pthread_nsg_attach, 13
- pthread_nsg_detach, 13
- pthread_nsg_get, 13
- pthread_num_ldoms_np, 9
- pthread_num_processors_np, 9
- pthread_processor_bind_np, 10
- pthread_pset_bind_np, 10
- pthread_rad_attach, 13
- pthread_rad_bind, 13
- PTHREAD_SCOPE_BOUND_NP, 12
- PTHREAD_SCOPE_SYSTEM, 12
- pthread_setaffinity_np, 15
- pthread_setrunon_np, 12
- pthread_use_only_cpu, 13

- ra_attachrset, 11
- ra_exec, 11
- ra_fork, 11
- rad_attach_pid, 13
- rad_bind_pid, 13
- rad_fork, 13
- rad_get_cpus, 13
- rad_get_current_home, 13
- rad_get_freemem, 13
- rad_get_max, 13
- rad_get_num, 13
- rad_get_physmem, 13
- rad_get_state, 13
- radaddset, 13
- radandset, 13
- radcopyset, 13
- radcountset, 13
- raddelset, 13

raddiffset, 13
rademptyset, 13
radfillset, 13
radisemptyset, 13
radismember, 13
radorset, 13
radsetcreate, 13
radsetdestroy, 13
radxorset, 13
RQMODE_ADVISORY, 12
rs_alloc, 11
rs_getassociativity, 10
rs_getinfo, 10
rs_getnamedrset, 11
rs_getrad, 11
rs_init, 11
rs_numrads, 10
rs_op, 11
rs_setnameattr, 11
rs_setpartition, 11
rtl_getcpuid, 15
runon, 15

sched_getaffinity, 14
sched_setaffinity, 14
SetProcessAffinityMask, 8
SetThreadAffinityMask, 8
SetThreadIdealProcessor, 8
SHM_LGPAGE, 11
SHM_PIN, 11
SHM_SHARE_MMU, 9
shmat, 9
shmget, 10, 11
SIGCPUFAIL, 11
SIGRECONFIG, 11
SMP, 4
sysconf, 8–10, 13, 14, 16
sysctl, 16
sysfs, 14
sysmp, 11
SYSPAGE_ENTRY, 15

td_ta_map_lwp2thr, 15
td_thr_get_info, 15
THREAD_BASIC_INFORMATION, 8
thread_self, 11
ThreadBasicInformation, 8
ThreadCtl, 15
ThreadCtl.r, 15

vmtune, 11

Windows, 7