High Performance Computing: Concepts, Methods, & Means

Operating Systems

Prof. Thomas Sterling
Department of Computer Science
Louisiana State University
March 22, 2007
Topics

- Introduction
- Overview of OS Roles and Responsibilities
- OS Concepts
- Unix family of OS
- Linux
- Lightweight Kernels
- Summary – Material for the Test
Topics

• Introduction
• Overview of OS Roles and Responsibilities
• OS Concepts
• Unix family of OS
• Linux
• Lightweight Kernels
• Summary – Material for the Test
Opening Remarks

• Last time: scheduling of work on system nodes
• But – what controls the nodes?
• Today: the Operating System
Topics

- Introduction
- **Overview of OS Roles and Responsibilities**
- OS Concepts
- Unix family of OS
- Linux
- Lightweight Kernels
- Summary – Material for the Test
What is an Operating System?
- A program that **controls the execution of application programs**
- An interface between applications and hardware

Primary functionality
- Exploits the hardware resources of one or more processors
- Provides a set of services to system users
- Manages secondary memory and I/O devices

Objectives
- **Convenience**: Makes the computer more convenient to use
- **Efficiency**: Allows computer system resources to be used in an efficient manner
- **Ability to evolve**: Permit effective development, testing, and introduction of new system functions without interfering with service

Services Provided by the OS

- Program development
  - Editors and debuggers
- Program execution
- Access to I/O devices
- Controlled access to files
- System access
- Error detection and response
  - Internal and external hardware errors
  - Software errors
  - Operating system cannot grant request of application
- Accounting
Layers of Computer System

Figure 2.1  Layers and Views of a Computer System
Resources Managed by the OS

- Processor
- Main Memory
  - volatile
  - referred to as real memory or primary memory
- I/O modules
  - secondary memory devices
  - communications equipment
  - terminals
- System bus
  - communication among processors, memory, and I/O modules
Topics

• Introduction
• Overview of OS Roles and Responsibilities
• OS Concepts
• Unix family of OS
• Linux
• Lightweight Kernels
• Summary – Material for the Test
Key OS Concepts

- Process management
- Memory management
- Storage management
- Information protection and security
- Scheduling and resource management
- System structure
A process is a program in execution. It is a unit of work within the system. Program is a passive entity, process is an active entity.

Process needs resources to accomplish its task
- CPU, memory, I/O, files
- Initialization data

Process termination requires reclaim of any reusable resources

Single-threaded process has one program counter specifying location of next instruction to execute
- Process executes instructions sequentially, one at a time, until completion

Multi-threaded process has one program counter per thread

Typically system has many processes, some user, some operating system running concurrently on one or more CPUs
- Concurrency by multiplexing the CPUs among the processes / threads
Process Management Activities

The operating system is responsible for the following activities in connection with process management:

- Creating and deleting both user and system processes
- Suspending and resuming processes
- Providing mechanisms for process synchronization
- Providing mechanisms for process communication
- Providing mechanisms for deadlock handling
Process Management & Scheduling

Process Management Support Structures

Process Scheduling
Multiprogramming & Multitasking

- **Multiprogramming** needed for efficiency
  - Single user cannot keep CPU and I/O devices busy at all times
  - Multiprogramming organizes jobs (code and data) so CPU always has one to execute
  - A subset of total jobs in system is kept in memory
  - One job selected and run via **job scheduling**
  - When it has to wait (for I/O for example), OS switches to another job

- **Timesharing (multitasking)** is logical extension in which CPU switches jobs so frequently that users can interact with each job while it is running, creating **interactive** computing
  - **Response time** should be < 1 second
  - Each user has at least one program executing in memory
    - **process**
  - If several jobs ready to run at the same time ⇒ **CPU scheduling**
  - If processes don’t fit in memory, **swapping** moves them in and out to run
  - **Virtual memory** allows execution of processes not completely in memory
Multiprogramming and Multiprocessing

(a) Interleaving (multiprogramming; one processor)

(b) Interleaving and overlapping (multiprocessing; two processors)

Figure 2.12 Multiprogramming and Multiprocessing
Memory Management

- All data in memory before and after processing
- All instructions in memory in order to execute
- Memory management determines what is in memory when
  - Optimizing CPU utilization and computer response to users
- Memory management activities
  - Keeping track of which parts of memory are currently being used and by whom
  - Deciding which processes (or parts thereof) and data to move into and out of memory
  - Allocating and deallocating memory space as needed
Virtual Memory

• Virtual Memory:
  – Allows programmers to address memory from a logical point of view
  – No hiatus between the execution of successive processes while one process was written out to secondary store and the successor process was read in

• Virtual Memory & File System:
  – Implements long-term store
  – Information stored in named objects called files

• Paging:
  – Allows process to be comprised of a number of fixed-size blocks, called pages
  – Virtual address is a page number and an offset within the page
  – Each page may be located any where in main memory
  – Real address or physical address in main memory
Translation Lookaside Buffer

Figure 8.7 Use of a Translation Lookaside Buffer
Figure 8.3  Address Translation in a Paging System
Storage Management

• OS provides uniform, logical view of information storage
  – Abstracts physical properties to logical storage unit - file
  – Each medium is controlled by device (i.e., disk drive, tape drive)
    • Varying properties include access speed, capacity, data-transfer rate, access method (sequential or random)

• File-System management
  – Files usually organized into directories
  – Access control on most systems to determine who can access what
  – OS activities include
    • Creating and deleting files and directories
    • Primitives to manipulate files and dirs
    • Mapping files onto secondary storage
    • Backup files onto stable (non-volatile) storage media
Protection and Security

- **Protection** – any mechanism for controlling access of processes or users to resources defined by the OS
- **Security** – defense of the system against internal and external attacks
  - Huge range, including denial-of-service, worms, viruses, identity theft, theft of service
- Systems generally first distinguish among users, to determine who can do what
  - User identities (**user IDs**, security IDs) include name and associated number, one per user
  - User ID then associated with all files, processes of that user to determine access control
  - Group identifier (**group ID**) allows set of users to be defined and controls managed, then also associated with each process, file
  - **Privilege escalation** allows user to change to effective ID with more rights
OS Kernel

• Kernel:
  – Portion of operating system that is in main memory
  – Contains most frequently used functions
  – Also called the nucleus

• Hardware Features:
  – Memory protection: Do not allow the memory area containing the monitor to be altered
  – Timer: Prevents a job from monopolizing the system
  – Privileged instructions: Certain machine level instructions can only be executed by the monitor
  – Interrupts: Early computer models did not have this capability

• Memory Protection
  – User program executes in user mode
    • Certain instructions may not be executed
  – Monitor executes in system mode
    • Kernel mode
    • Privileged instructions are executed
    • Protected areas of memory may be accessed
Scheduling and Resource Management

• Fairness
  – Give equal and fair access to resources

• Differential responsiveness
  – Discriminate among different classes of jobs

• Efficiency
  – Maximize throughput, minimize response time, and accommodate as many uses as possible
Modern Operating Systems

- Small operating system core
- Contains only essential core operating systems functions
- Many services traditionally included in the operating system are now external subsystems
  - Device drivers
  - File systems
  - Virtual memory manager
  - Windowing system
  - Security services
- Microkernel architecture
  - Assigns only a few essential functions to the kernel
    - Address spaces
    - Interprocess communication (IPC)
    - Basic scheduling
Benefits of a Microkernel Organization

- Uniform interface on request made by a process
  - Don’t distinguish between kernel-level and user-level services
  - All services are provided by means of message passing
- Extensibility
  - Allows the addition of new services
- Flexibility
  - New features added & existing features can be subtracted
- Portability
  - Changes needed to port the system affect only the microkernel itself
- Reliability
  - Modular design
  - Small microkernel can be rigorously tested
- Distributed system support
  - Message are sent without knowing what the target machine is
- Object-oriented operating system
  - Uses components with clearly defined interfaces (objects)
Monolithic OS vs. Microkernel

(a) Layered kernel
(b) Microkernel

Figure 4.10  Kernel Architecture
Modern Operating Systems

• Multithreading
  – Process is divided into threads that can run concurrently
    • Thread
      – Dispatchable unit of work
      – executes sequentially and is interruptable
    • Process is a collection of one or more threads

• Symmetric multiprocessing (SMP)
  – There are multiple processors
  – These processors share same main memory and I/O facilities
  – All processors can perform the same functions

• Distributed operating systems
  – Provides the illusion of a single main memory space and single secondary memory space

• Object-oriented design
  – Used for adding modular extensions to a small kernel
  – Enables programmers to customize an operating system without disrupting system integrity
Thread and SMP Management

Example: Solaris Multithreaded Architecture
Topics

- Introduction
- Overview of OS Roles and Responsibilities
- OS Concepts
- Unix family of OS
- Linux
- Lightweight Kernels
- Summary – Material for the Test
Brief History of UNIX

- Initially developed at Bell Labs in late 1960s by a group including Ken Thompson, Dennis Ritchie and Douglas McIlroy
- Originally named *Unics* in contrast to *Multics*, a novel experimental OS at the time
- The first deployment platform was PDP-7 in 1970
- Rewritten in C in 1973 to enable portability to other machines (most notably PDP-11) – an unusual strategy as most OS’s were written in assembly language
- Version 6 (version numbers were determined by editions of system manuals), released in 1976, was the first widely available version outside Bell Labs
- Version 7 (1978) is the ancestor of most modern UNIX systems
- The most important non-AT&T implementation is UNIX BSD, developed at the UC at Berkeley and to run on PDP and VAX
- By 1982 Bell Labs combined various UNIX variants into a single system, marketed as UNIX System III, which later evolved into System V
Traditional UNIX Organization

- Hardware is surrounded by the operating system software
- Operating system is called the system kernel
- Comes with a number of user services and interfaces
  - Shell
  - Components of the C compiler
UNIX Process Management

• Nine process states (see the next slide)
  – Two Running states (kernel and user)
  – Process running in kernel mode cannot be preempted (hence no real-time processing support)

• Process description
  – User-level context: basic elements of user’s program, generated directly from compiled object file
  – Register context: process status information, stored when process is not running
  – System-level context: remaining information, contains static and dynamic part

• Process control
  – New processes are created via fork() system call, in which kernel:
    • Allocates a slot in the process table,
    • Assigns a unique ID to the new process,
    • Obtains a copy of the parent process image,
    • Increment counters for files owned by the parent,
    • Changes state of the new process to Ready to Run,
    • Returns new process ID to the parent process, and 0 to the child
## Description of Process States

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Running</td>
<td>Executing in user mode.</td>
</tr>
<tr>
<td>Kernel Running</td>
<td>Executing in kernel mode.</td>
</tr>
<tr>
<td>Ready to Run, in Memory</td>
<td>Ready to run as soon as the kernel schedules it.</td>
</tr>
<tr>
<td>Asleep in Memory</td>
<td>Unable to execute until an event occurs; process is in main memory (a blocked state).</td>
</tr>
<tr>
<td>Ready to Run, Swapped</td>
<td>Process is ready to run, but the swapper must swap the process into main memory before the kernel can schedule it to execute.</td>
</tr>
<tr>
<td>Sleeping, Swapped</td>
<td>The process is awaiting an event and has been swapped to secondary storage (a blocked state).</td>
</tr>
<tr>
<td>Preempted</td>
<td>Process is returning from kernel to user mode, but the kernel preempts it and does a process switch to schedule another process.</td>
</tr>
<tr>
<td>Created</td>
<td>Process is newly created and not yet ready to run.</td>
</tr>
<tr>
<td>Zombie</td>
<td>Process no longer exists, but it leaves a record for its parent process to collect.</td>
</tr>
</tbody>
</table>
Process State Transition Diagram

- User Running
  - return to user
  - system call, interrupt
- Kernel Running
  - preempt
  - reschedule process
- Zombie
  - exit
  - interrupt, interrupt return
- Preempted
- Ready to Run In Memory
  - reschedule process
  - sleep
- Created
  - fork
  - enough memory
  - not enough memory (swapping system only)
- Ready to Run Swapped
  - swap out
  - swap in
- Asleep in Memory
  - wakeup
  - swap out
- Sleep, Swapped
  - wakeup

CENTER FOR COMPUTATION & TECHNOLOGY AT LOUISIANA STATE UNIVERSITY
# UNIX Process

## User-Level Context
- **Process text**: Executable machine instructions of the program.
- **Process data**: Data accessible by the program of this process.
- **User stack**: Contains the arguments, local variables, and pointers for functions executing in user mode.
- **Shared memory**: Memory shared with other processes, used for interprocess communication.

## Register Context
- **Program counter**: Address of next instruction to be executed; may be in kernel or user memory space of this process.
- **Processor status register**: Contains the hardware status at the time of preemption; contents and format are hardware dependent.
- **Stack pointer**: Points to the top of the kernel or user stack, depending on the mode of operation at the time or preemption.
- **General-purpose registers**: Hardware dependent.

## System-Level Context
- **Process table entry**: Defines state of a process; this information is always accessible to the operating system.
- **U (user) area**: Process control information that needs to be accessed only in the context of the process.
- **Per process region table**: Defines the mapping from virtual to physical addresses; also contains a permission field that indicates the type of access allowed the process: read-only, read-write, or read-execute.
- **Kernel stack**: Contains the stack frame of kernel procedures as the process executes in kernel mode.

## UNIX Process Structure

![UNIX Process Structure Diagram]

<table>
<thead>
<tr>
<th>Process ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>User IDs</td>
</tr>
<tr>
<td>Signal Dispatch Table</td>
</tr>
<tr>
<td>Memory Map</td>
</tr>
<tr>
<td>Priority</td>
</tr>
<tr>
<td>Signal Mask</td>
</tr>
<tr>
<td>Registers</td>
</tr>
<tr>
<td>STACK</td>
</tr>
<tr>
<td>File Descriptors</td>
</tr>
<tr>
<td>Processor State</td>
</tr>
</tbody>
</table>
UNIX Concurrency Mechanisms

- **Pipes**
  - Circular buffers allowing two processes to communicate using producer-consumer model

- **Messages**
  - Rely on `msgsnd` and `msgrcv` primitives
  - Each process has a message queue acting as a mailbox

- **Shared memory**
  - Fastest communication method
  - Block of shared memory may be accessed by multiple processes

- **Semaphores**
  - Synchronize processes’ access to resources

- **Signals**
  - Inform of the occurrences of asynchronous events
Traditional UNIX Scheduling

- Multilevel feedback using round-robin within each of the priority queues
- One-second preemption
- Priority calculation given by (recomputed once per second):

\[
CPU_j(i) = \frac{CPU_j(i-1)}{2}
\]

\[
P_j(i) = Base_j + \frac{CPU_j(i)}{2} + nice_j
\]

where

- \(CPU_j(i)\) = Measure of processor utilization by process \(j\) through interval \(i\)
- \(P_j(i)\) = Priority of process \(j\) at beginning of interval \(i\); lower values equal higher priorities
- \(Base_j\) = Base priority of process \(j\)
- \(nice_j\) = user-controllable adjustment factor
SVR4 “two-handed clock” policy:

- Each swappable page has a reference bit in page table entry
- The bit is cleared when the page is first brought in
- The bit is set when the page is referenced
- The fronthand sets the reference bits to zero as it sweeps through the list of pages
- Sometime later, the backhand checks reference bits; if a bit is zero, the page is added to page-out candidate list
I/O classes in UNIX:

- **Buffered (data pass through system buffers)**
  - System buffer caches
    - Managed using three lists: free list, device list and driver I/O queue
    - Follows readers/writers model
    - Serve block-oriented devices (disks, tapes)
  - Character queues
    - Serve character-oriented devices (terminals, printers, …)
    - Use producer-consumer model
- **Unbuffered (typically involving DMA between the I/O module and process I/O area)**
UNIX File Types

• Regular
  – Contains arbitrary data stored in zero or more data blocks
  – Treated as stream of bytes by the system

• Directory
  – Contains list of file names along with pointers to associated nodes (index nodes, or inodes)
  – Organized in hierarchies

• Special
  – Contains no data, but serves as a mapping to physical devices
  – Each I/O device is associated with a special file

• Named pipe
  – Implement inter-process communication facility in file system name space

• Link
  – Provides name aliasing mechanism for files

• Symbolic link
  – Data file containing the name of file it is linked to
Directory Structure and File Layout

Inode table

Directory

i1  Name1
i2  Name2
i3  Name3
i4  Name4

...
Modern UNIX Systems

• System V Release 4 (SVR4)
  – Developed jointly by AT&T and Sun Microsystems
  – Improved, feature-rich and most widespread rewrite of System V

• Solaris 10
  – Developed by Sun Microsystems, based on SVR4

• 4.4BSD
  – Released by Berkeley Software Distribution
  – Used as a basis of a number of commercial UNIX products (e.g. Mac OS X)

• Linux
  – Discussed in detail later
Modern UNIX Kernel

Source: U. Vahalia “UNIX Internals: The New Frontiers”
Topics

- Introduction
- Overview of OS Roles and Responsibilities
- OS Concepts
- Unix family of OS
- Linux
- Lightweight Kernels
- Summary – Material for the Test
Linux History

• Initial version written by Linus Torvalds (Finland) in 1991
• Originally intended as a non-commercial replacement for the Minix kernel
• Since then, a number of contributors continued to improve Linux collaborating over the Internet under Torvalds’ control:
  – Added many features available in commercial counterparts
  – Optimized the performance
  – Ported to other hardware architectures (Intel x86 and IA-64, IBM Power, MIPS, SPARC, ARM and others)
• The source code is available and free (protected by the GNU Public License)
• The current kernel version is 2.6.20
• Today Linux can be found on plethora of computing platforms, from embedded microcontrollers and handhelds, through desktops and workstations, to servers and supercomputers
Linux Design

• Monolithic OS
  – All functionality stored mainly in a single block of code
  – All components of the kernel have access to all internal data structures and routines
  – Changes require relinking and frequently a reboot

• Modular architecture
  – Extensions of kernel functionality (modules) can be loaded and unloaded at runtime (dynamic linking)
  – Can be arranged hierarchically (stackable)
  – Overcomes use and development difficulties associated with monolithic structure
Principal Kernel Components

- Signals
- System calls
- Processes and scheduler
- Virtual memory
- File systems
- Network protocols
- Character device drivers
- Block device drivers
- Network device drivers
- Traps and faults
- Physical memory
- Interrupts
Linux Kernel Components

[Diagram showing various components of the Linux kernel, including signals, system calls, processes & scheduler, virtual memory, char device drivers, file systems, block device drivers, network protocols, network device drivers, traps & faults, physical memory, interrupts, CPU, system memory, terminal, disk, and network interface controller.]
Linux Process Components

- State (running, ready, suspended, stopped, zombie)
- Scheduling information
- Identifiers (PID, user and group)
- Interprocess communications (SysV primitives)
- Links (parent process, siblings and children)
- Times and timers
- File system usage (open files, current and root dir.)
- Address space
- Processor-specific context (registers and stack)
Linux Concurrency Mechanisms

- Atomic operations on data
  - Integer (access an integer variable)
  - Bitmap (operate on one bit in a bitmap)

- Spinlocks: protect critical sections
  - Basic: plain (when code does not affect interrupt state), _irq (interrupts are always enabled), _irqsave (it is not known if the interrupts are enabled), _bh (“bottom half”; the minimum of work is performed by the interrupt handler)
  - Reader-writer: allows multiple threads to access the same data structure

- Semaphores: support interface from UNIX SVR4
  - Binary
  - Counting
  - Reader-writer

- Barriers: enforce the order of memory updates
Linux Memory Management

• Virtual memory addressing
  – Page directory (occupies one page per process)
  – Page middle directory (possibly multiple pages; each entry points to one page in the page table)
  – Page table (spans possibly multiple pages; each entry refers to one virtual page of the process)

• Page allocation
  – Based on the “clock” algorithm
  – 8-bit age variable instead of “use” bit (LFU policy)

• Kernel memory allocation
  – Main memory page frames (for user-space processes, dynamic kernel data, kernel code and page cache)
  – Slab allocation for smaller chunks
Linux Virtual Address Translation

Virtual address

- Global Directory
- Middle Directory
- Page Table
- Offset

- Page directory
- Page middle directory
- Page table

- cr3 register

Page frame in physical memory
Linux Scheduling

• Real-time scheduling since version 2.4
• Scheduling classes
  – SCHED_FIFO (FIFO real-time)
  – SCHED_RR (Round Robin real-time)
  – SCHED_OTHER (non real-time)
• Multiple priorities within each of the classes
• O(1) scheduler for non-real time threads
  – Time to select the thread and assign it to a processor is constant, regardless of the load
  – Separate queue for each priority level
  – All queue organized in two structures: an active queues structure and an expired queues structure
Linux O(1) Scheduler

Active Queues:
140 queues by priority; each queue contains ready tasks for that priority

Expired Queues:
140 queues by priority; each queue contains ready tasks with expired time slices for that priority
Linux I/O

• Disk scheduling
  – Linus Elevator: maintains single sorted queue of requests
  – Deadline: three queues (sorted elevator, read queue and write queue); associates expiration time with each request
  – Anticipatory: superimposed on the deadline scheduler; attempts to merge successive requests accessing neighboring blocks
  – CFQ, or “Completely Fair Queueing”: based on the anticipatory scheduler; attempts to divide the bandwidth of device fairly among all accessing it processes

• Page cache
  – Originally separate cache for regular FS access and virtual memory pages, and a buffer cache for block I/O
  – Since version 2.4, the page cache is unified for all traffic between disk and main memory
Linux Virtual File System

• Designed to support a variety of file management systems and file structures
  – Assumes that files are objects that share basic properties (symbolic names, ownership, access protection, etc.)
  – The functionality limited to small set of operations: create, read, write, delete, …
  – Mapping module needed to transform the characteristics of a real FS to that expected by VFS

• VFS objects:
  – Superblock: represents mounted FS
  – Inode: represents a file
  – Dentry: represents directory entry
  – File object: represents an open file associated with a process
Linux TCP/IP Stack
Topics

- Introduction
- Overview of OS Roles and Responsibilities
- OS Concepts
- Unix family of OS
- Linux
- Lightweight Kernels
- Summary – Material for the Test
Heterogeneous nodes:

- **Compute (BG/L specific)**
  - Run specialized OS supporting computations efficiently
- **I/O (BG/L specific)**
  - Use OS flexibly supporting various forms of I/O
- **Service (generic)**
  - Uses conventional off-the-shelf OS
  - Provides support for the execution of compute and I/O node operating systems
- **Front-end (generic)**
  - Support program compilation, submission and debugging
- **File server (generic)**
  - Store data that the I/O nodes read and write

BG/L Processing Sets

• Processing Set (pset): a logical entity combining one I/O node with a collection of compute nodes
  – Supported number of compute nodes in a pset ranges from 8 to 128 (in powers of 2)
• Every system partition is organized as a collection of psets
  – All psets in a partition must have the same number of compute nodes
  – The psets of a partition must cover all I/O and compute nodes in the partition, but may not overlap
• Arranged to reflect the topological proximity between I/O and compute nodes in order to
  – Improve the communication performance and scalability within pset by exploiting regularity
  – Simplify software stack
Software Stack in Compute Node

- CNK controls all access to hardware, and enables bypass for application use
- User-space libraries and applications can directly access torus and tree through bypass
- As a policy, user-space code should not directly touch hardware, but there is no enforcement of that policy

Compute Node Memory Map

- Memory mapped devices
  - Torus, tree, lockbox, sram

- Application memory
  - Executable, heap, stack, mapped to largest TLB
  - DDR scratchpad
  - CNK
  - 256K code, 256K local0, 256K local1, 256K shared

Compute Node Kernel (CNK)

- Lean Linux-like kernel (fits in 1MB of memory)
- The primary goal is to “stay out of way and let the application run”
- Performs job startup sequence on every node of a partition
  - Creates address space for execution of compute processes
  - Loads code and initialized data for the executable
  - Transfers processor control to the loaded executable
- Memory management
  - Address spaces are flat and fixed (no paging), and fit statically into PowerPC 440 TLBs
  - Two scenarios supported (assuming 512MB/node option):
    - Coprocessor mode: one 511 MB address space for single process
    - Virtual node mode: two 255 MB spaces for two processes
- No process scheduling: only one thread per processor
- Processor control stays within the application, unless:
  - The application issues a system call
  - Timer interrupt is received (requested by the application code)
  - An abnormal event is detected, requiring kernel’s attention
• Compute Node Kernel supports
  – 68 Linux system calls (file I/O, directory operations, signals, process information, time, sockets)
  – 18 CNK-specific calls (cache manipulation, SRAM and DRAM management, machine and job information, special-purpose register access)

• System call scenarios
  – **Simple** calls requiring little OS functionality (e.g. accessing timing register) are handled locally
  – **I/O** calls using file system infrastructure or IP stack are shipped for execution in the I/O node associated with the issuing compute node
  – **Unsupported** calls requiring infrastructure not supported in BG/L (e.g. `fork()` or `mmap()`) return immediately with error condition
I/O Node Functionality

• Executes embedded version of Linux
  – No swap space, in-memory root file system, absence of most daemons and services
  – Full TCP/IP stack
  – File system support, with available ports of GPFS, Lustre, NFS, PVFS2

• Plays dual role in the system
  – Master of the corresponding pset
    • Initializes job launch on compute nodes in its partition
    • Loads and starts application code on each processor in the pset
    • Never runs actual application processes
  – Server for requests issued by compute nodes in a pset
    • Runs Control and I/O Daemon (CIOD) to link the compute processes of an application to the outside world

• Benefits
  – Compute node OS may be very simple
  – Minimal of interference between computation and I/O
  – Avoids security and safety issues (no need for daemons to clean up after misbehaving jobs)
Function Shipping from CNK to CIOD

- CIOD processes requests from
  - Control system using socket to the service node
  - Debug server using a pipe to a local process
  - Compute nodes using the tree network

- I/O system call sequence:
  - CNK trap
  - Call parameters are packaged and sent to CIOD in the corresponding I/O node
  - CIOD unpacks the message and reissues it to Linux kernel on I/O node
  - After call completes, the results are sent back to the requesting CNK (and the application)
Service Node Overview

• Runs BG/L control software, responsible for operation and monitoring of compute and I/O nodes
• Sets up BG/L partitions and loads initial state and code in the partition nodes (they are stateless)
• Isolates the partition from others in the system
• Computes routing for torus, collective and global interrupt networks
• Instantiates compute and I/O node personalities:

<table>
<thead>
<tr>
<th>Compute node personality</th>
<th>I/O node personality</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Memory size</td>
<td>• Memory size</td>
</tr>
<tr>
<td>• Bit-steering for memory configuration</td>
<td>• Bit-steering for memory configuration</td>
</tr>
<tr>
<td>• Physical location</td>
<td>• Physical location</td>
</tr>
<tr>
<td>• (x, y, z) torus coordinates</td>
<td>• Pset number and size</td>
</tr>
<tr>
<td>• Pset number and size</td>
<td>• Routes for the collectives network</td>
</tr>
<tr>
<td>• Routes for the collectives network</td>
<td>• MAC and IP address</td>
</tr>
<tr>
<td></td>
<td>• Broadcast and gateway IP addresses</td>
</tr>
<tr>
<td></td>
<td>• Service node IP address</td>
</tr>
<tr>
<td></td>
<td>• NFS server address and export directory</td>
</tr>
<tr>
<td></td>
<td>• Security key</td>
</tr>
</tbody>
</table>
Sandia/UNM Lightweight Kernel (LWK) Design Goals

- Targeted at massively parallel environments comprised of thousands of processors with distributed memory and a tightly coupled network
- Provide necessary support for scalable, performance-oriented scientific applications
- Offer a suitable development environment for parallel applications and libraries
- Emphasize efficiency over functionality
- Maximize the amount of resources (e.g. CPU, memory, and network bandwidth) allocated to the application
- Seek to minimize time to completion for the application
- Provide deterministic performance
LWK Approach

• Separate policy decision from policy enforcement
• Move resource management decisions as close to application as possible
  – Applications always know how to manage resources better than the operating system
• Protect applications from each other
  – Requirement in a classified computing environment
• Get out of the way
LWK General Structure

PCT

App. 1

libc.a

libmpi.a

App. 2

libc.a

libmpi.a

App. 3

libc.a

libmpi.a

QK
Quintessential Kernel (QK)

• Policy enforcer
• Initializes hardware
• Handles interrupts and exceptions
• Maintains hardware virtual address tables
• Fixed size
  – No dependence on the size of the system or the size of the parallel job
• Small number of well-defined, non-blocking entry points
• No virtual memory support
Process Control Thread (PCT)

- Privileged user-level process
- Policy maker
  - Process loading (with yod)
  - Process scheduling
  - Virtual address space management
  - Fault handling
  - Signals
- Designed to allow for customizing OS policies
  - Single-tasking or multi-tasking
  - Round-robin or priority scheduling
  - High-performance, debugging, or profiling version
- Changes behavior of OS without changing the kernel
Yod

- Parallel job launcher
  - Communicates with PCTs to provide scalable broadcast of executables and shell environment

- Runs in the service partition

- Services standard I/O and system call requests from compute node processes once job is running
  - System calls such as open() are forwarded to yod via a remote procedure call mechanism
LWK Key Ideas

• Protection
  – Levels of trust

• Kernel is small
  – Very reliable

• Kernel is static
  – No structures depend on how many processes are running

• Resource management pushed out to application processes and runtime system

• Services pushed out of kernel to PCT and runtime system
Topics

- Introduction
- Overview of OS Roles and Responsibilities
- OS Concepts
- Unix family of OS
- Linux
- Lightweight Kernels
- **Summary – Material for the Test**
Summary – Material for the Test

- Definition, services provided (slides 6, 7)
- OS concepts: process management (slide 14)
- Multitasking & multiprogramming (slide 16)
- OS concepts: memory management (slide 18, 19)
- OS concepts: protection & security (slide 23)
- Benefits of microkernel (slide 27)
- Unix process state transition (slides 36, 37)
- Unix concurrency mechanisms (slide 39)
- SVR4 page replacement strategy (slide 41)
- Linux kernel component (slides 50-54)
- Linux scheduling (slides 57, 58)
- Compute Node Kernel BG/L (slides 69, 70)
References

- Maurice Bach "The Design of the UNIX Operating System"
- Stallings "official" slides based on the book (one pdf per chapter; most useful are sections of chapters 2, 3, 4, 7, 8, 9 and 12):
- Stallings shortened notes on UNIX
  – http://www.box.net/public/tjoikg2scz
- and Linux:
  – http://www.box.net/public/xg654evf8u
- J. Moreira et al. paper on BG/L OS design: