CSMC 412

Operating Systems
Prof. Ashok K Agrawala

Set 12

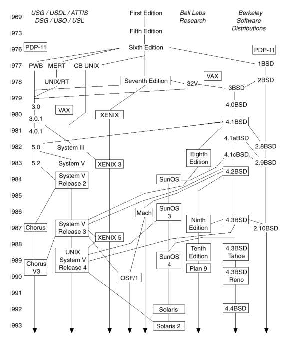
The Unix System

- Process Management
- Memory Management

History

- First developed in 1969 by Ken Thompson and Dennis Ritchie of the Research Group at Bell Laboratories; incorporated features of other operating systems, especially MULTICS.
- The third version was written in C, which was developed at Bell Labs specifically to support UNIX.
- The most influential of the non-Bell Labs and non-AT&T UNIX development groups — University of California at Berkeley (Berkeley Software Distributions).
  - 4BSD UNIX resulted from DARPA funding to develop a standard UNIX system for government use.
  - Developed for the VAX, 4.3BSD is one of the most influential versions, and has been ported to many other platforms.
- Several standardization projects seek to consolidate the variant flavors of UNIX leading to one programming interface to UNIX.

History of UNIX Versions



CPU Scheduling

- Every process has a *scheduling priority* associated with it; larger numbers indicate lower priority.
- Negative feedback in CPU scheduling makes it difficult for a single process to take all the CPU time.
- Process aging is employed to prevent starvation.
- When a process chooses to relinquish the CPU, it goes to *sleep* on an *event*.
- When that event occurs, the system process that knows about it calls *wakeup* with the address corresponding to the event, and *all* processes that had done a *sleep* on the same address are put in the ready queue to be run.

Memory Management

- The initial memory management schemes were constrained in size by the relatively small memory resources of the PDP machines on which UNIX was developed.
- Pre 3BSD system use swapping exclusively to handle memory contention among processes: If there is too much contention, processes are swapped out until enough memory is available.
- Allocation of both main memory and swap space is done first-fit.

Memory Management (Cont.)

- Sharable text segments do not need to be swapped; results in less swap traffic and reduces the amount of main memory required for multiple processes using the same text segment.
- The *scheduler process* (or *swapper*) decides which processes to swap in or out, considering such factors as time idle, time in or out of main memory, size, etc.
- In 4.3BSD, swap space is allocated in pieces that are multiples of power of 2 and minimum size, up to a maximum size determined by the size or the swap-space partition on the disk.

Paging

- Berkeley UNIX systems depend primarily on paging for memory-contention management, and depend only secondarily on swapping.
- *Demand paging* – When a process needs a page and the page is not there, a page fault tot he kernel occurs, a frame of main memory is allocated, and the proper disk page is read into the frame.
- A *pagedaemon* process uses a modified second-chance page-replacement algorithm to keep enough free frames to support the executing processes.
- If the scheduler decides that the paging system is overloaded, processes will be swapped out whole until the overload is relieved.

The Linux System

- Process Management
- Scheduling
- Memory Management

History

- Linux is a modem, free operating system based on UNIX standards.
- First developed as a small but self-contained kernel in 1991 by Linus Torvalds, with the major design goal of UNIX compatibility.
- Its history has been one of collaboration by many users from all around the world, corresponding almost exclusively over the Internet.
- It has been designed to run efficiently and reliably on common PC hardware, but also runs on a variety of other platforms.
- The core Linux operating system kernel is entirely original, but it can run much existing free UNIX software, resulting in an entire UNIX-compatible operating system free from proprietary code.

Process Scheduling

- Linux uses two process-scheduling algorithms:
  - A time-sharing algorithm for fair preemptive scheduling between multiple processes
  - A real-time algorithm for tasks where absolute priorities are more important than fairness
- A process's scheduling class defines which algorithm to apply.
- For time-sharing processes, Linux uses a prioritized, credit based algorithm.
  - The crediting rule

    $$\text{credits} := \frac{\text{credits}}{2} + \text{priority}$$

    factors in both the process's history and its priority.
  - This crediting system automatically prioritizes interactive or I/O-bound processes.

Process Scheduling (Cont.)

- Linux implements the FIFO and round-robin real-time scheduling classes; in both cases, each process has a priority in addition to its scheduling class.
  - The scheduler runs the process with the highest priority; for equal-priority processes, it runs the longest-waiting one
  - FIFO processes continue to run until they either exit or block
  - A round-robin process will be preempted after a while and moved to the end of the scheduling queue, so that round-robing processes of equal priority automatically time-share between themselves.

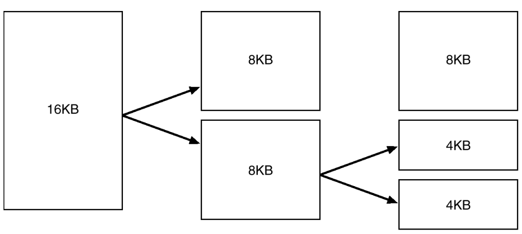
Symmetric Multiprocessing

- Linux 2.0 was the first Linux kernel to support SMP hardware; separate processes or threads can execute in parallel on separate processors.
- To preserve the kernel's non-preemptible synchronization requirements, SMP imposes the restriction, via a single kernel spinlock, that only one processor at a time may execute kernel-mode code.

Memory Management

- Linux's physical memory-management system deals with allocating and freeing pages, groups of pages, and small blocks of memory.
- It has additional mechanisms for handling virtual memory, memory mapped into the address space of running processes.

Splitting of Memory in a Buddy Heap



Managing Physical Memory

- The page allocator allocates and frees all physical pages; it can allocate ranges of physically-contiguous pages on request.
- The allocator uses a *buddy-heap* algorithm to keep track of available physical pages.
  - Each allocatable memory region is paired with an adjacent partner.
  - Whenever two allocated partner regions are both freed up they are combined to form a larger region.
  - If a small memory request cannot be satisfied by allocating an existing small free region, then a larger free region will be subdivided into two partners to satisfy the request.
- Memory allocations in the Linux kernel occur either statically (drivers reserve a contiguous area of memory during system boot time) or dynamically (via the page allocator).

Virtual Memory

- The VM system maintains the address space visible to each process: It creates pages of virtual memory on demand, and manages the loading of those pages from disk or their swapping back out to disk as required.
- The VM manager maintains two separate views of a process's address space:
  - A logical view describing instructions concerning the layout of the address space.
    The address space consists of a set of nonoverlapping regions, each representing a continuous, page-aligned subset of the address space.
  - A physical view of each address space which is stored in the hardware page tables for the process.

Virtual Memory (Cont.)

- Virtual memory regions are characterized by:
  - The backing store, which describes from where the pages for a region come; regions are usually backed by a file or by nothing (*demand-zero* memory)
  - The region's reaction to writes (page sharing or copy-on-write).
- The kernel creates a new virtual address space
  1. When a process runs a new program with the exec system call
  2. Upon creation of a new process by the fork system call

Virtual Memory (Cont.)

- On executing a new program, the process is given a new, completely empty virtual-address space; the program-loading routines populate the address space with virtual-memory regions.
- Creating a new process with fork involves creating a complete copy of the existing process's virtual address space.
  - The kernel copies the parent process's VMA descriptors, then creates a new set of page tables for the child.
  - The parent's page tables are copies directly into the child's, with the reference count of each page covered being incremented.
  - After the fork, the parent and child share the same physical pages of memory in their address spaces.

Virtual Memory (Cont.)

- The VM paging system relocates pages of memory from physical memory out to disk when the memory is needed for something else.
- The VM paging system can be divided into two sections:
  - The pageout-policy algorithm decides which pages to write out to disk, and when.
  - The paging mechanism actually carries out the transfer, and pages data back into physical memory as needed.

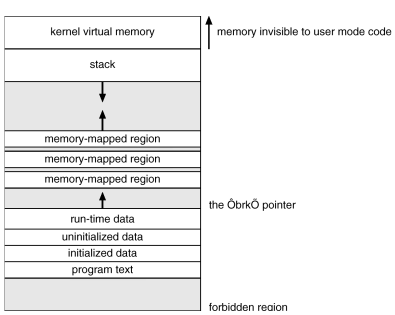
Virtual Memory (Cont.)

- The Linux kernel reserves a constant, architecture-dependent region of the virtual address space of every process for its own internal use.
- This kernel virtual-memory area contains two regions:
  - A static area that contains page table references to every available physical page of memory in the system, so that there is a simple translation from physical to virtual addresses when running kernel code.
  - The reminder of the reserved section is not reserved for any specific purpose; its page-table entries can be modified to point to any other areas of memory.

Executing and Loading User Programs

- Linux maintains a table of functions for loading programs; it gives each function the opportunity to try loading the given file when an exec system call is made.
- The registration of multiple loader routines allows Linux to support both the ELF and a.out binary formats.
- Initially, binary-file pages are mapped into virtual memory; only when a program tries to access a given page will a page fault result in that page being loaded into physical memory.
- An ELF-format binary file consists of a header followed by several page-aligned sections; the ELF loader works by reading the header and mapping the sections of the file into separate regions of virtual memory.

Memory Layout for ELF Programs



Static and Dynamic Linking

- A program whose necessary library functions are embedded directly in the program's executable binary file is *statically* linked to its libraries.
- The main disadvantage of static linkage is that every program generated must contain copies of exactly the same common system library functions.
- *Dynamic* linking is more efficient in terms of both physical memory and disk-space usage because it loads the system libraries into memory only once.

Windows NT

- History
- Design Principles
- Memory Management

Windows NT

- 32-bit preemptive multitasking operating system for modern microprocessors.
- Key goals for the system:
  - portability
  - security
  - POSIX compliance
  - multiprocessor support
  - extensibility
  - international support
  - compatibility with MS-DOS and MS-Windows applications.
- Uses a micro-kernel architecture.
- Available in two versions, Windows NT Workstation and Windows NT Server.
- In 1996, more NT server licenses were sold than UNIX licenses

History

- In 1988, Microsoft decided to develop a “new technology” (NT) portable operating system that supported both the OS/2 and POSIX APIs.
- Originally, NT was supposed to use the OS/2 API as its native environment but during development NT was changed t use the Win32 API, reflecting the popularity of Windows 3.0.

Design Principles

- Extensibility — layered architecture.
  - NT executive, which runs in protected mode, provides the basic system services.
  - On top of the executive, several server subsystems operate in user mode.
  - Modular structure allows additional environmental subsystems to be added without affecting the executive.
- Portability — NT can be moved from on hardware architecture to another with relatively few changes.
  - Written in C and C++.
  - Processor-dependent code is isolated in a dynamic link library (DLL) called the “hardware abstraction layer” (HAL).

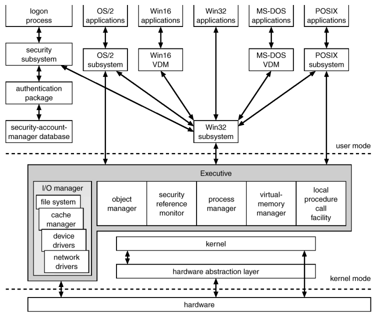
Design Principles (Cont.)

- Reliability — NT uses hardware protection for virtual memory, and software protection mechanisms for operating system resources.
- Compatibility — applications that follow the IEEE 1003.1 (POSIX) standard can be complied to run on NT without changing the source code.
- Performance — NT subsystems can communicate with one another via high-performance message passing.
  - Preemption of low priority threads enables the system to respond quickly to external events.
  - Designed for symmetrical multiprocessing.
- International support — supports different locales via the national language support (NLS) API.

NT Architecture

- Layered system of modules.
- Protected mode — HAL, kernel, executive.
- User mode — collection of subsystems
  - Environmental subsystems emulate different operating systems.
  - Protection subsystems provide security functions.

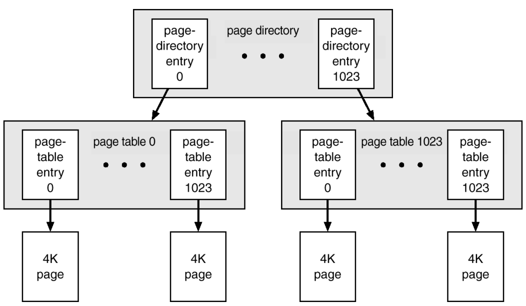
Depiction of NT Architecture



Executive — Virtual Memory Manager

- **The design of the VM manager assumes that the underlying hardware supports virtual to physical mapping a paging mechanism, transparent cache coherence on multiprocessor systems, and virtual addressing aliasing.**
- **The VM manager in NT uses a page-based management scheme with a page size of 4 KB.**
- **The NT manager uses a two step process to allocate memory.**
  - **The first step reserves a portion of the process's address space.**
  - **The second step commits the allocation by assigning space in the NT paging file.**

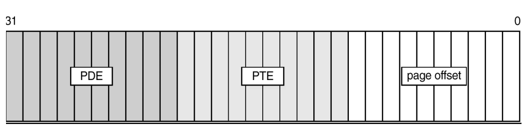
Virtual-Memory Layout



Virtual Memory Manager (Cont.)

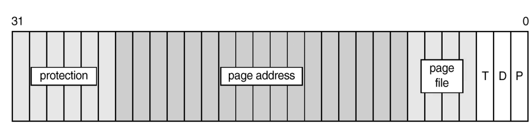
- **The virtual address translation in NT uses several data structures.**
  - **Each process has a *page directory* that contains 1024 *page directory entries* of size 4 bytes.**
  - **Each page directory entry points to a *page table* which contains 1024 *page table entries* (PTEs) of size 4 bytes.**
  - **Each PTE points to a 4 KB *page frame* in physical memory.**
- **A 10-bit integer can represent all the values form 0 to 1023, therefore, can select any entry in the page directory, or in a page table.**
- **This property is used when translating a virtual address pointer to a bye address in physical memory.**
- **A page can be in one of six states: valid, zeroed, free standby, modified and bad.**

The PTE Structure



- **5 bits for page protection, 20 bits for page frame address, 4 bits to select a paging file, and 3 bits that describe the page state.**

Standard Page-Table Entry



T—transition
D—dirty
P—present