What is Linux?

- 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
- Developed in collaboration by many users around the world over Internet
- Designed to run efficiently and reliably on common PC hardware, but also runs on a variety of other platforms (e.g., PDAs)
- 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
- In use by over 12 million users

Linux Kernel History

- Version 0.01 (May 1991) — no networking, ran only on x86 hardware, limited device-drive support, supported only the Minix file system
- Version 1.0 (March 1994) — TCP/IP networking, IPC, enhanced file system, floppy-disk & CDROM support, paging
- Version 1.2 (March 1995) — some support for SPARC, Alpha, & MIPS architectures
- Version 2.0 (June 1996) — multiple architectures, multiprocessor support
- Current version is Version 2.5.x

Linux Distribution

- Standard, precompiled sets of packages, or distributions, include the basic Linux system, system installation and management utilities, and ready-to-install packages of common UNIX tools
- The first distributions managed these packages by simply providing a means of unpacking all the files into the appropriate places; modern distributions include advanced package management
- Early distributions included SLS and Slackware. Red Hat and Debian are popular distributions from commercial and noncommercial sources, respectively
- The RPM Package file format permits compatibility among the various Linux distributions

Linux Licensing

- Linux is distributed under GNU General Public License (GPL)
  - Linux is not public domain — *public domain* implies that the author(s) have relinquished copyright over their software
  - Linux is not shareware — *shareware* implies that the authors are distributing their software to users as try-and-buy and expect to be paid
- The GPL allows software to be freely used and modified by anybody.
  - The GPL is written such that the none of the freely available code can be turned into commercial (or closed) product, although a reasonable distribution fee can be charged
  - The GPL stipulates that the source code must always distributed together with compiled binaries

Linux Kernel Modules

- Kernel code executes in kernel mode with full access to all the physical resources of the computer
- Kernel modules are pieces of kernel code that can be independently compiled, loaded, and unloaded
  - A kernel module may typically implement a device driver, a file system, or a networking protocol
  - Third parties can write and distribute, on their own terms, device drivers or file systems that could not be distributed under the GPL
- Module management allows modules to be dynamically loaded into memory when needed
  - Regularly queries the kernel, and will unload modules no longer actively used

Linux Process Management

- Two distinct operations (like UNIX):
  - **fork** — creates a new process
  - **execve** — runs a new program
- Under Linux, process properties fall into three groups:
  - Process identity — process ID, process credentials (user ID & group ID for file and resource access), personality (!)
  - Process environment — command-line arguments, environment variables
  - Process context — state of running program, including:
    - Scheduling context
    - Accounting information
    - File table & file-system context
    - Virtual-memory context

Linux Processes and Threads

- Same internal representation for processes and threads; a thread is simply a new process that shares the same address space as its parent.
- Only distinction is when a new thread is created by the **clone** system call:
  - **fork** creates a new process with its own entirely new process context
  - **clone** creates a new process (thread) with its own identity, but that is allowed to share the data structures of its parent
- Create new process / thread => create new identity and scheduling contexts
  - Fork (a process) => copy other contexts
  - Clone (a thread) => share other contexts

Linux Kernel Synchronization

- Kernel-mode execution is requested:
  - By a user program who requests an OS service, either explicitly via a system call, or implicitly, e.g., a page fault
  - By a device driver through a hardware interrupt, causing a kernel-mode interrupt handler to be invoked
- The kernel's critical sections are protected and run without interruption:
  - Normal kernel code is nonpreemptible
  - Interrupts are disabled during a critical section in the interrupt handler
    - To avoid having interrupts disabled for a long time, interrupt handlers are divided
    - Top half is a normal interrupt handler, can be interrupted by higher-priority processes
    - Bottom half is run, with interrupts enabled, by a miniature scheduler that ensures that bottom halves never interrupt themselves

Linux Process Scheduling

- For time-sharing processes, Linux uses a prioritized, credit based algorithm
  - The process with the greatest number of credits is selected
  - A process loses a credit at every timer interrupt
  - When credits reach 0 the process is suspended
  - Processes gain credits as they age according to this rule
    - The crediting rule
      credits = credits/2 + priority
      considers process's history and priority
  - This crediting system automatically prioritizes interactive or I/O-bound processes, since CPU-bound processes exhaust their credits quickly

Linux Process Scheduling (cont.)

- For real-time scheduling, Linux implements FIFO and round-robin; 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

Linux Memory Management

- The page 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
- 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 Linux Ext2fs File System

- Ext2fs uses a mechanism similar to that of BSD Fast File System (ffs) for locating data blocks belonging to a specific file, differing in their disk allocation policies:
  - In ffs, the disk is allocated to files in blocks of 8KB, with blocks being subdivided into fragments of 1KB to store small files or partially filled blocks at the end of a file
  - Ext2fs does not use fragments; it performs its allocations in smaller units. The default block size on ext2fs is 1KB, although 2KB and 4KB blocks are also supported
  - Ext2fs uses allocation policies designed to place logically adjacent blocks of a file into physically adjacent blocks on disk, so that it can submit an I/O request for several disk blocks as a single operation

The Linux Ext2fs File System (cont.)