Chapter 18: The Linux System

Zhi Wang
Florida State University

Content

- Linux history
- Design principles
- Kernel modules
- Process management
- Scheduling
- Memory management
- File systems
- Input and output
- Inter-process communication
- Network structure

Objectives

- To explore the history of the UNIX operating system from which Linux is derived and the principles which Linux is designed upon
- To examine the Linux **process model** and illustrate how Linux **schedules** processes and provides interprocess communication
- To look at **memory management** in Linux
- To explore how Linux implements **file systems** and manages I/O devices

History

- Linux is a modern, 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
- Many, varying Linux Distributions including the kernel, applications, and management tools

The Linux Kernel

- Version 0.01 was released at May 1991
  - no networking
  - ran only on 80386-compatible Intel processors and on PC hardware
  - extremely limited device-drive support
  - supported only the Minix file system
- Linux 1.0 (March 1994) included these new features:
  - support UNIX's standard TCP/IP networking protocols
  - BSD-compatible socket interface for networking programming
  - device-driver support for running IP over an Ethernet
  - enhanced file system
  - support for a range of SCSI controllers for high-performance disk access
  - extra hardware support
- Version 1.2 (March 1995) was the final PC-only Linux kernel

Linux 2.0

- Version 2.0 was released in June 1996
  - support for multiple architectures, including a fully 64-bit native Alpha port
  - support for multiprocessor architectures
  - improved memory-management code
  - improved TCP/IP performance
  - support for internal kernel threads
  - standardized configuration interface
- 2.4 and 2.6 increased SMP support
  - added journaling file system
  - preemptive kernel
  - 64-bit memory support

The Linux System

- Linux uses many free tools developed as part of
  - Berkeley’s BSD operating system
    - socket interface
    - networking tools (e.g., traceroute…)
  - MIT’s X Window System
  - Free Software Foundation's GNU project
    - bin-utilities, gcc, gnu libc…
- Linux used to developed by individual, now also big cooperators
  - IBM, Intel, Red hat, Marvell, Microsoft...
- Main Linux repository: www.kernel.org

Linux Distributions

- Standard, precompiled sets of packages, or distributions
  - include the basic Linux system
  - system installation and management utilities
  - ready-to-install packages of common UNIX tools
- Popular Linux distributions
  - Ubuntu, Fedora, Debian, Open Suse, …
  - see distrowatch.com

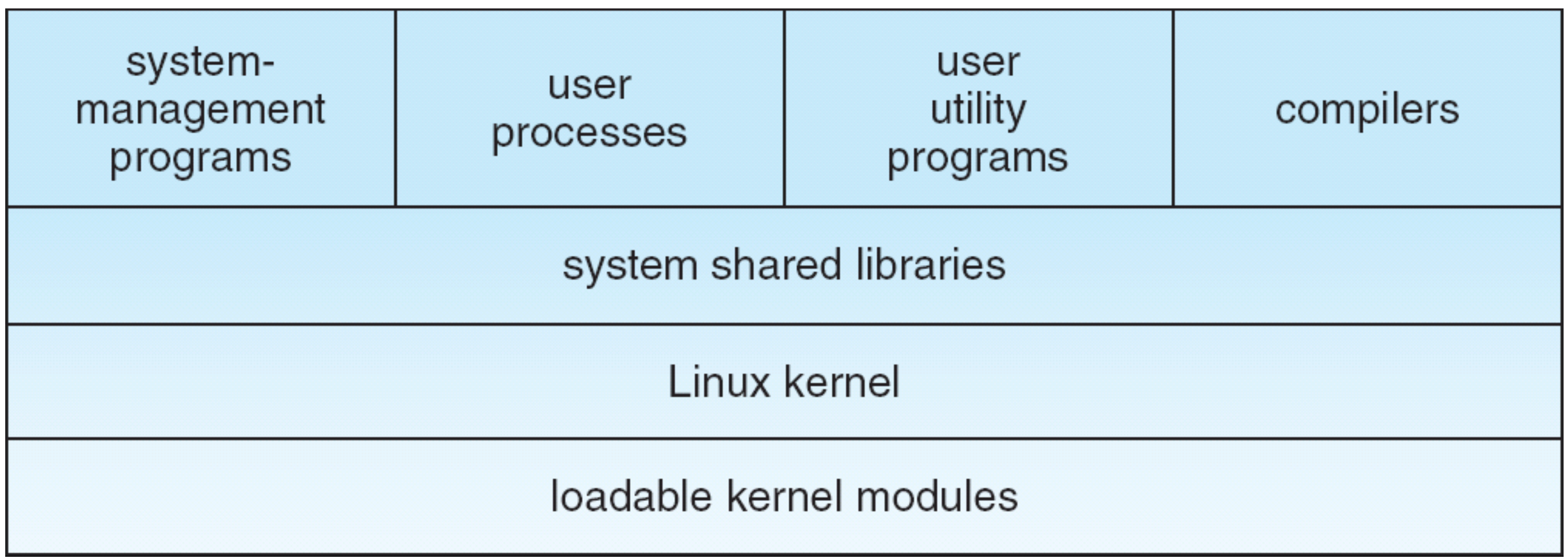
Linux Licensing

- Linux kernel is distributed under GNU General Public License (GPL)
  - GPL is defined by the Free Software Foundation
- GPL implications:
  - anyone using Linux, or creating their own derivative of Linux, may not make the derived (public) product proprietary
  - software released under GPL may not be redistributed as binary-only
- LGPL: Lesser GPL
  - allow non-(L)GPL software to link to LGPL licensed software

Design Principles

- Linux is a multiuser, multitasking system
- Linux is UNIX compatible
  - its file system adheres to traditional UNIX semantics
  - it fully implements the standard UNIX networking model
  - its API adheres to the SVR4 UNIX semantics
  - it is POSIX-compliant
- Linux supports a wide variety of architectures
- Main design goals are speed, efficiency, and standardization

Components of a Linux System



Kernel Modules

- Kernel code that can be compiled, loaded, and unloaded independently
  - it allows a Linux system to be set up with standard minimal kernel
    - other components loaded as modules
  - typically to implement device drivers, file systems, or networking protocols
- Three components to Linux module support:
  - module management
    - load/unload the module
    - resolve symbols (similar to a linker)
  - driver registration
    - kernel define an interface, module implement the interface
    - module registers to the kernel, kernel maintain a list of loaded modules
  - conflict resolution
    - resource conflicts
- Tools to support kernel modules: **lsmod**, **rmmod**, **modprobe**

Process Management

- Linux process management follows the Unix model:
  - **fork** system call creates a new process
  - a new program is run after a call to **execve**
- **Process control block** contains all the information about the process
  - process's identity
  - process environment
  - process context

Process Identity

- **Process ID** (PID): the unique identifier for the process
- **Credentials**: each process has an associated UID and group IDs
  - determine the process's rights to access system resources and files.
- **Personality**: the ABI the process conforms to
  - Linux supports ABIs of different flavors of UNIX (e.g., BSD)
  - personality selects the ABI used by the process
  - not traditionally found on UNIX systems

Process Environment

- Environment is inherited from its parent
  - argument vector lists the command-line arguments used to it
    - conventionally starts with the program name
  - environment vector is a list of “NAME=VALUE” pairs
    - associates named environment variables with arbitrary textual values
    - e.g., PATH=“/usr/bin;….”
- Environment variables can be used to pass data among processes
- Environment variables can be set per-process

Process Context

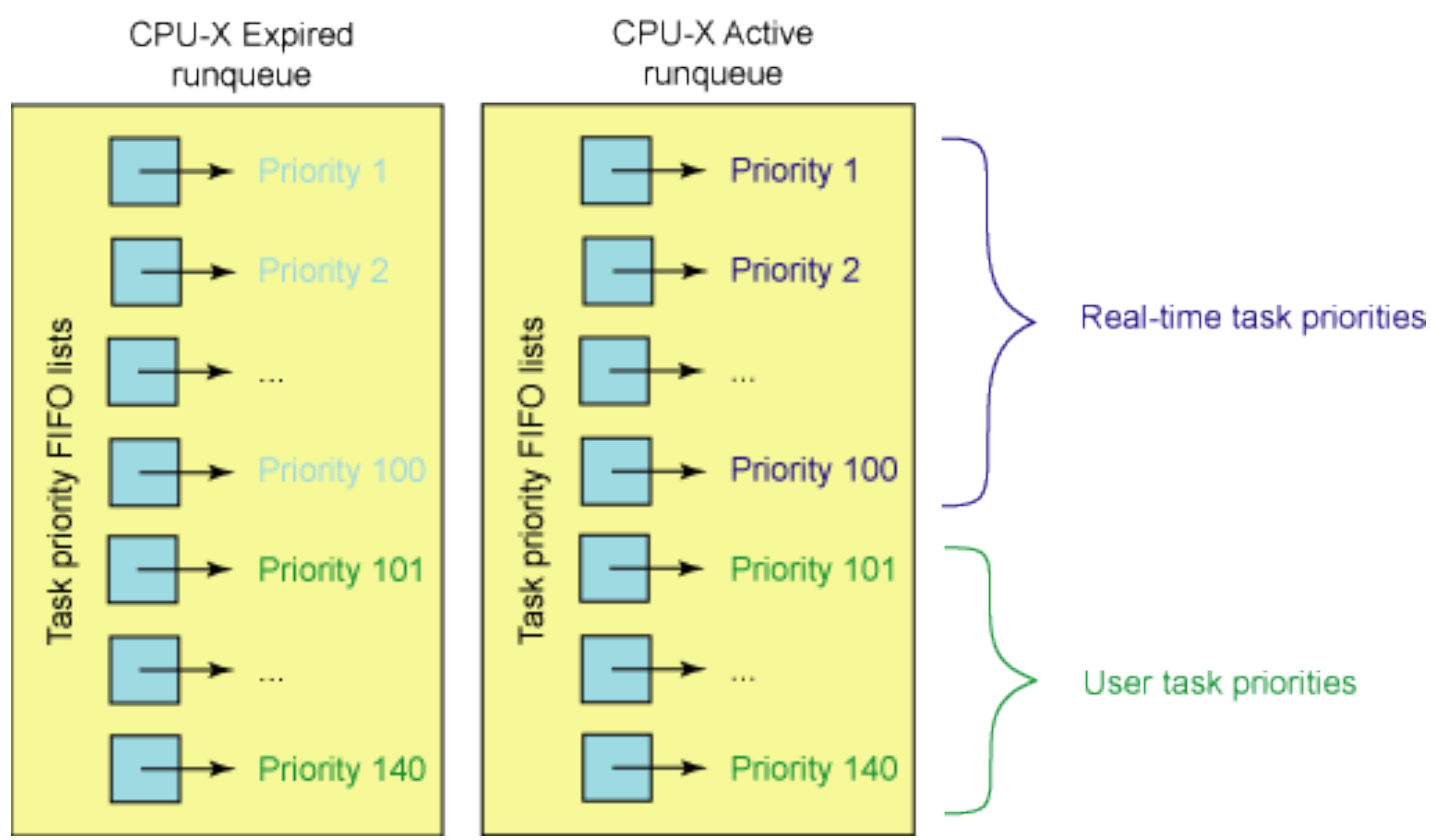
- (Constantly changing) state of a running program at any point in time.
  - many context information: resources, scheduling, files, accounting…
  - **scheduling context** is the most important part of the process context
  - **file table** is an array of pointers to kernel file structures.
    - when making file I/O, processes refer to files by their index into this table
  - **signal-handler** defines the routine to be called upon some events
  - **virtual memory** defines the process's address space
  - …

Processes and Threads

- Linux uses the same internal representation for processes and threads
  - a thread is a new process sharing the same address space as its parent.
- A distinction is made when a new thread is created by clone
  - **clone** allows fine-grained control over what is shared between two threads
  - **fork** creates a new process with its own entirely new process context

Scheduling

- Allocate CPU time to different tasks within an operating system
  - Linux supports both user processes and kernel tasks
  - kernel tasks may be requested by a running process, or executes internally on behalf of a device driver



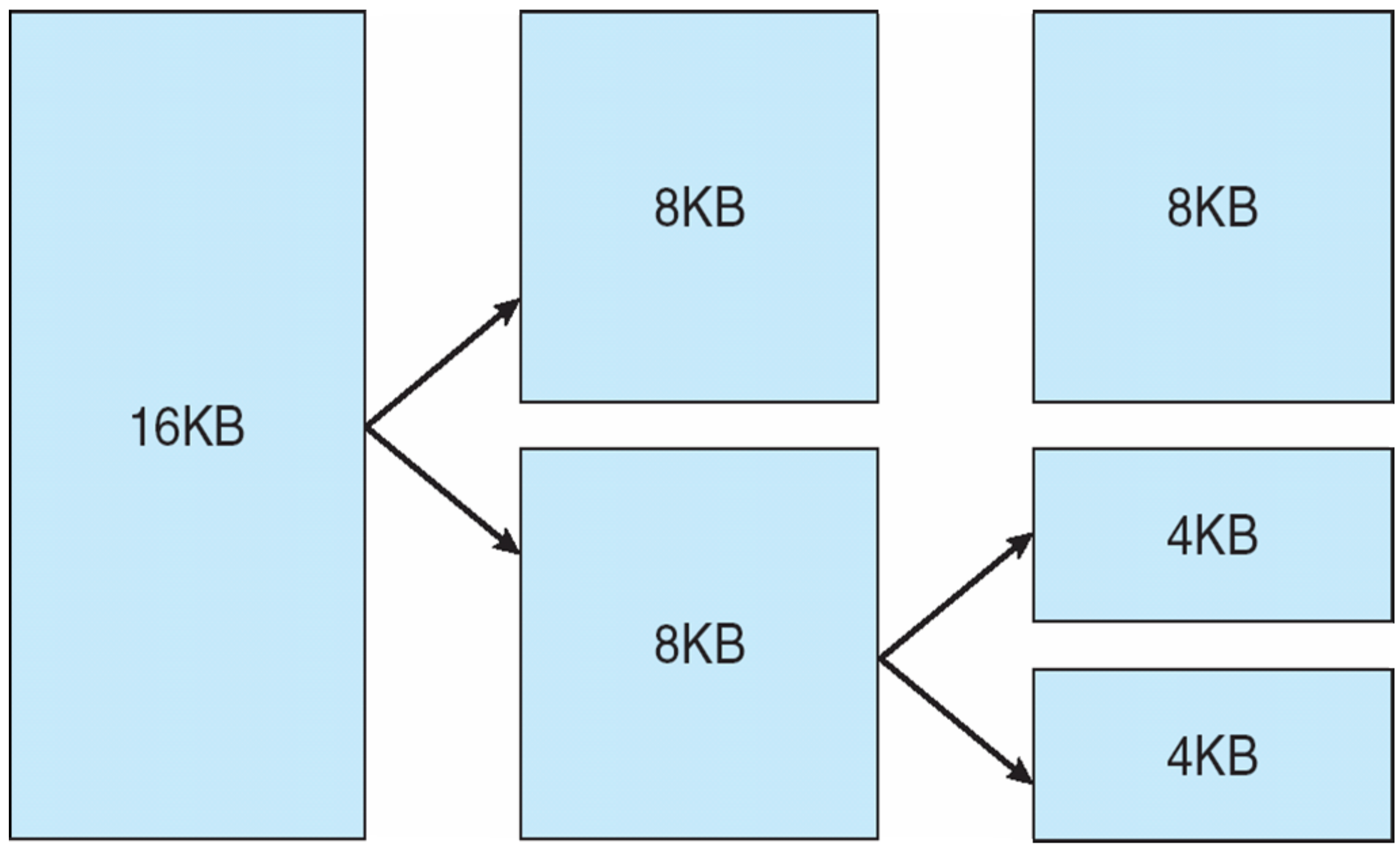
Memory Management

- Linux's physical MM system deals with allocating and freeing:
  - pages, groups of pages, and small blocks of memory
- Memory is split into 3 different **zones** due to hardware characteristics
- Two major types of allocator:
  - **page allocator** allocates physical pages using buddy algorithm
    - many data structure needs whole pages (e.g., driver buffers)
  - **slab allocator** allocates memory in smaller sizes (kernel objects)

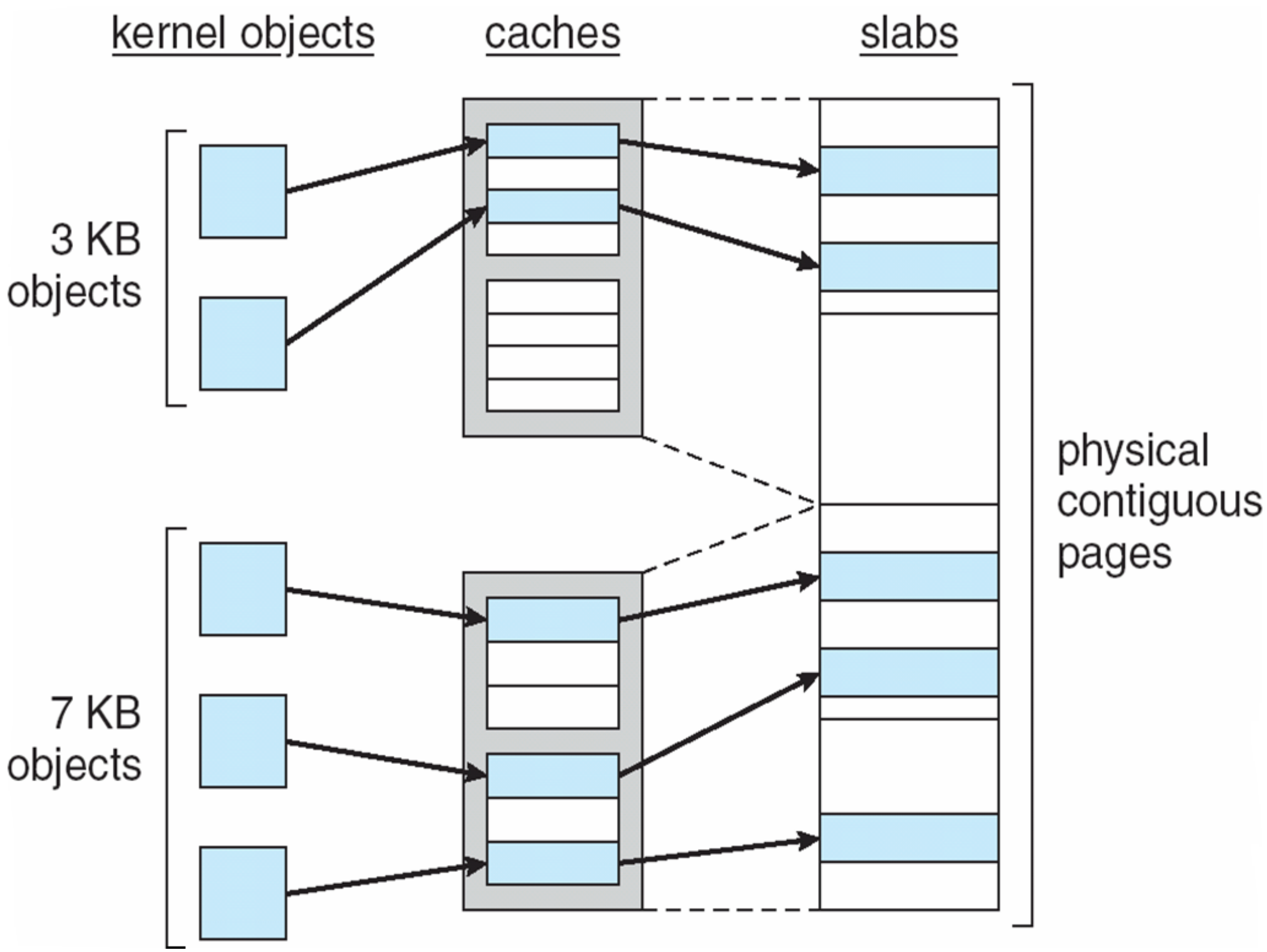
Memory Zones

zone	physical memory
ZONE_DMA	< 16 MB
ZONE_NORMAL	16 .. 896 MB
ZONE_HIGHMEM	> 896 MB

Page Allocator (Buddy System)



Slab Allocator



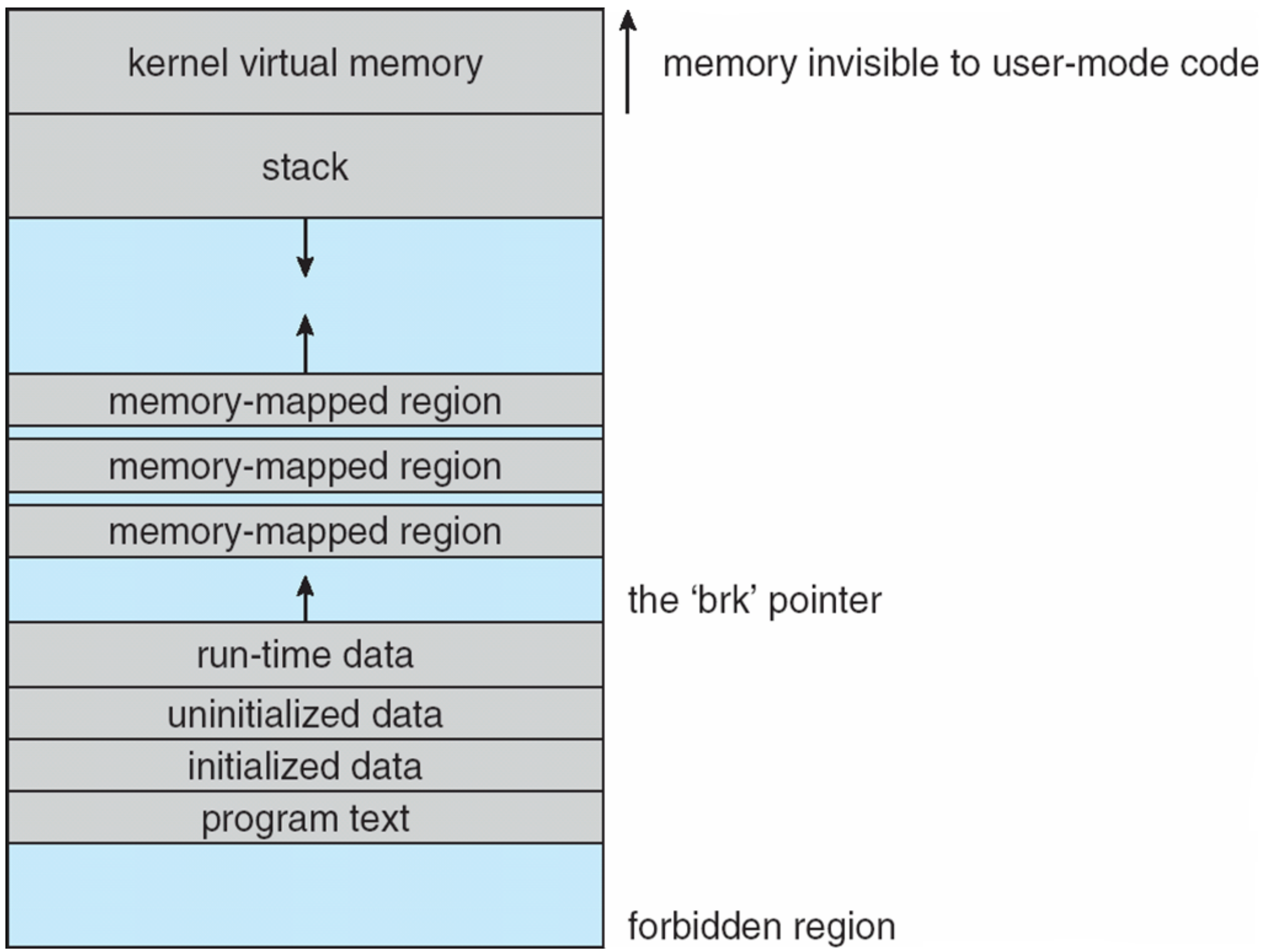
Virtual Memory

- VM maintains address space of each process
- Linux VM supports many different feature
  - demand paging, swapping, copy-on-write, memory mapped files…
- Linux keeps track of every physical page (**struct page**)
  - each physical frame has an associated struct page
    - keep struct page at minimal to avoid waste of memory

Executing and Loading User Programs

- Linux can load many different executable file formats
    - ELF, a.out…
    - ELF is the most command format
    - ELF file has a header followed by several page-aligned sections
- Binary files are loaded into memory using **demand paging**

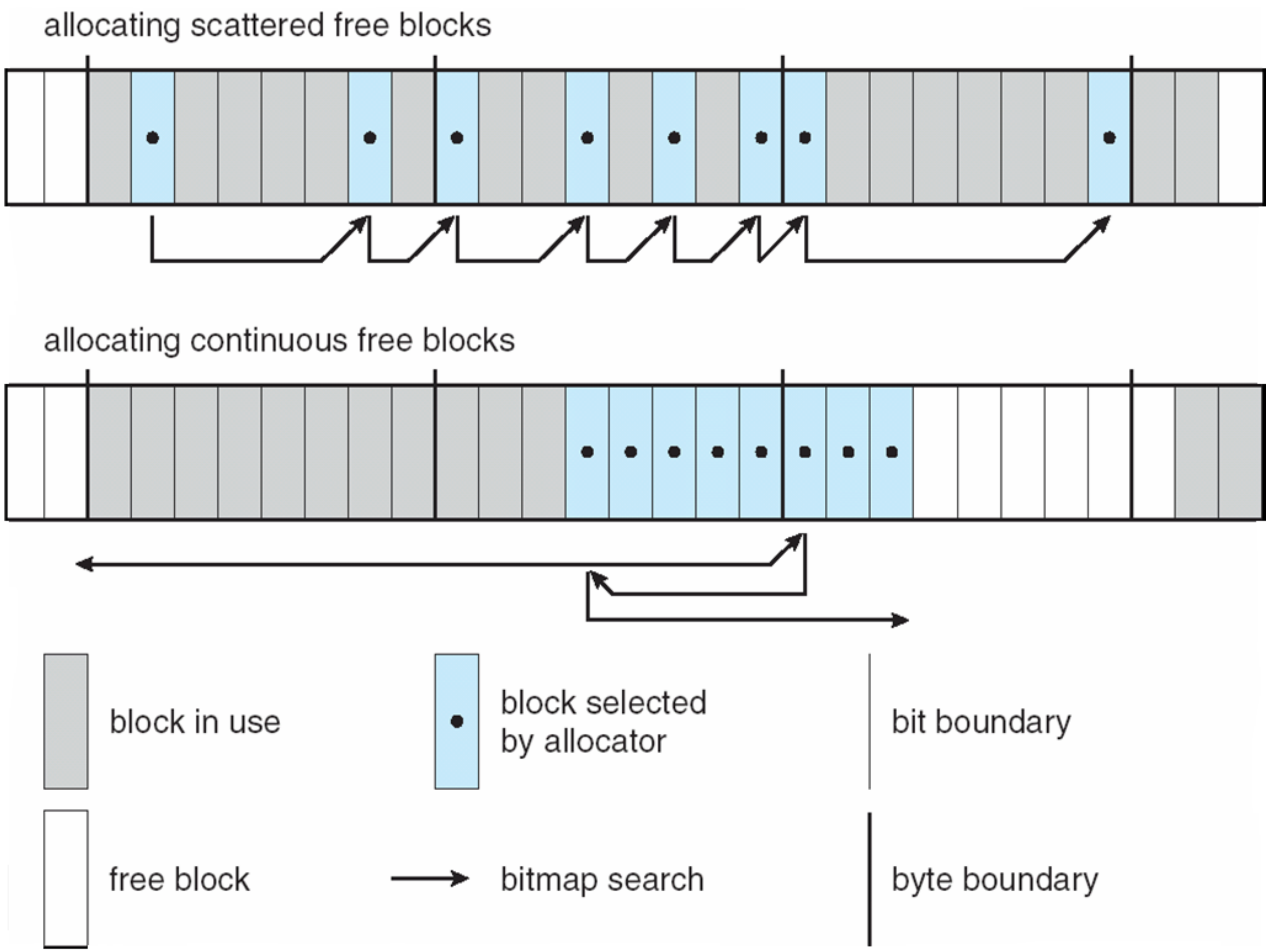
Memory Layout for ELF Programs



File Systems

- To the user, Linux's file system is a hierarchical directory tree
- Internally, Linux kernel use the **virtual file system** (VFS)
  - Linux supports many many file systems
    - Ext2, Ext3, Ext4, Btrfs, NTFS, FAT/FAT32, …
- Linux also supports synthetic file systems, such as the **/proc** file system
  - /proc does not store data, file content is computed on demand
  - /proc provides many statistics about the kernel

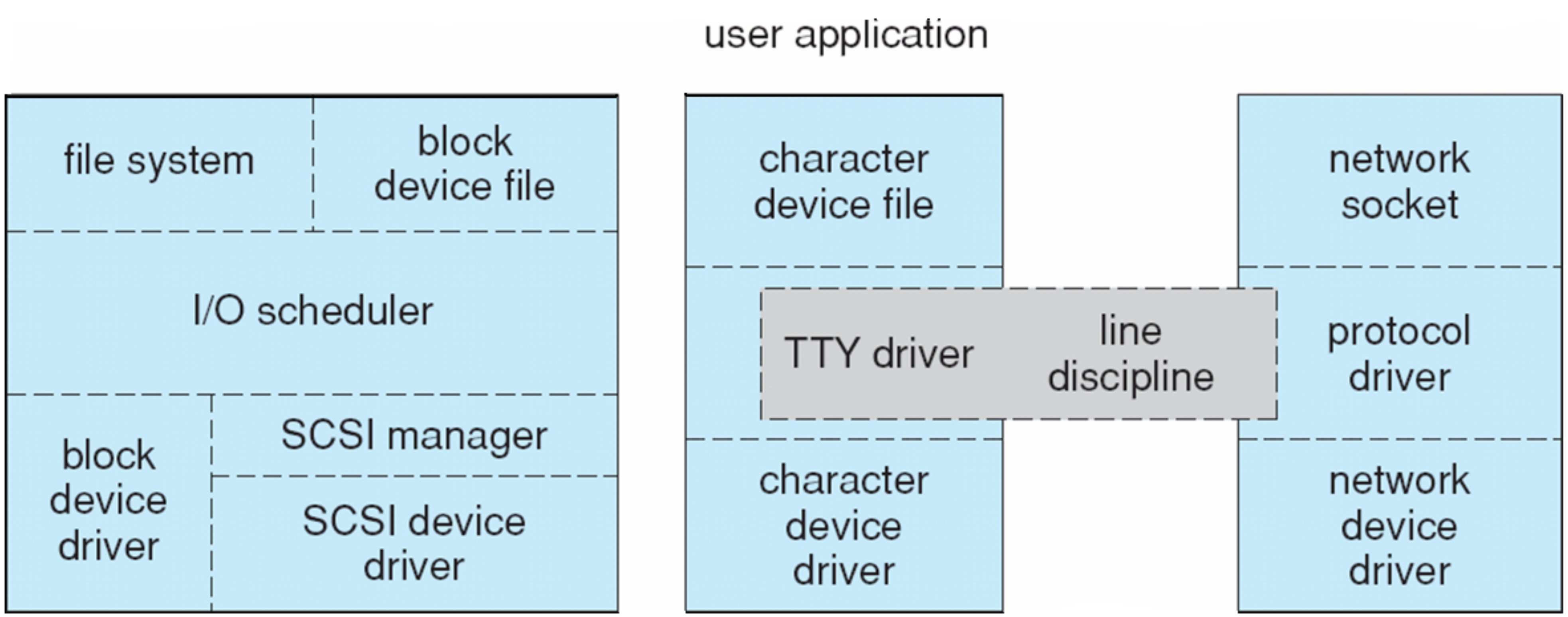
Ext2fs Block-Allocation (Bitmap)



Input and Output

- Linux device I/O uses two types of cache:
  - data is cached in the page cache
    - unified cache with the virtual memory system
  - metadata is cached in the buffer cache
    - a separate cache indexed by the physical disk block
- Linux splits all devices into three classes:
  - **block devices** allow random access to independent, fixed size blocks of data
  - **character devices** include most other devices
    - they don't need to support the functionality of regular files
  - **network devices** are interfaced via the kernel's networking subsystem

Device-Driver Block Structure



Inter-process Communication

- Linux informs processes that an event has occurred via signals
  - there is a limited number of signals
  - signals cannot carry information, only the fact that a signal has occurred
- Linux supports SVR4 IPC
  - pipe, shared memory, synchronization…

Network Structure

- Networking is a key area of functionality for Linux.
  - it supports the standard Internet protocols for UNIX to UNIX communications
  - It also implements protocols native to non-UNIX operating systems
    - e.g., Apple talk, IPX (Novell), Netbios
- Internally, Linux networking is implemented by three layers of software:
  - socket interface
  - protocol drivers
  - network device drivers

End of Chapter 21