OPERATING SYSTEMS

Storage Management

Venkatesh Prasad

Department of Computer Science

OPERATING SYSTEMS

Case Study: Unix/Linux File systems

Venkatesh Prasad

Department of Computer Science

OPERATING SYSTEMS

Slides Credits for all PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne - 9th edition 2013 and some slides from 10th edition 2018
2. Some conceptual text and diagram from Operating Systems - Internals and Design Principles, William Stallings, 9th edition 2018
3. Some presentation transcripts from A. Frank – P. Weisberg
4. Some conceptual text from Operating Systems: Three Easy Pieces, Remzi Arpaci-Dusseau, Andrea Arpaci Dusseau

- To the user, Linux’ s file system appears as a hierarchical directory tree obeying UNIX semantics
- Internally, the kernel hides implementation details and manages the multiple different file systems via an abstraction layer, that is, the virtual file system (VFS)
- The Linux VFS is designed around object-oriented principles and is composed of four components:
  - A set of definitions that define what a file object is allowed to look like
    - The **inode object** structure represent an individual file
    - The **file object** represents an open file
    - The **superblock object** represents an entire file system
    - A **dentry object** represents an individual directory entry

Linux File System (Cont.)

- The Linux VFS is designed around object-oriented principles and layer of software to manipulate those objects with a set of operations on the objects
  - For example for the file object operations include (from struct file_operations in /usr/include/linux/fs.h

    int open(. . .) — Open a file

    ssize t read(. . .) — Read from a file

    ssize t write(. . .) — Write to a file

    int mmap(. . .) — Memory-map a file

Linux File System (Cont.)

- The Linux VFS is designed around object-oriented principles and layer of software to manipulate those objects with a set of operations on the objects
  - For example for the file object operations include (from struct file_operations in /usr/include/linux/fs.h

    int open(. . .) — Open a file

    ssize t read(. . .) — Read from a file

    ssize t write(. . .) — Write to a file

    int mmap(. . .) — Memory-map a file

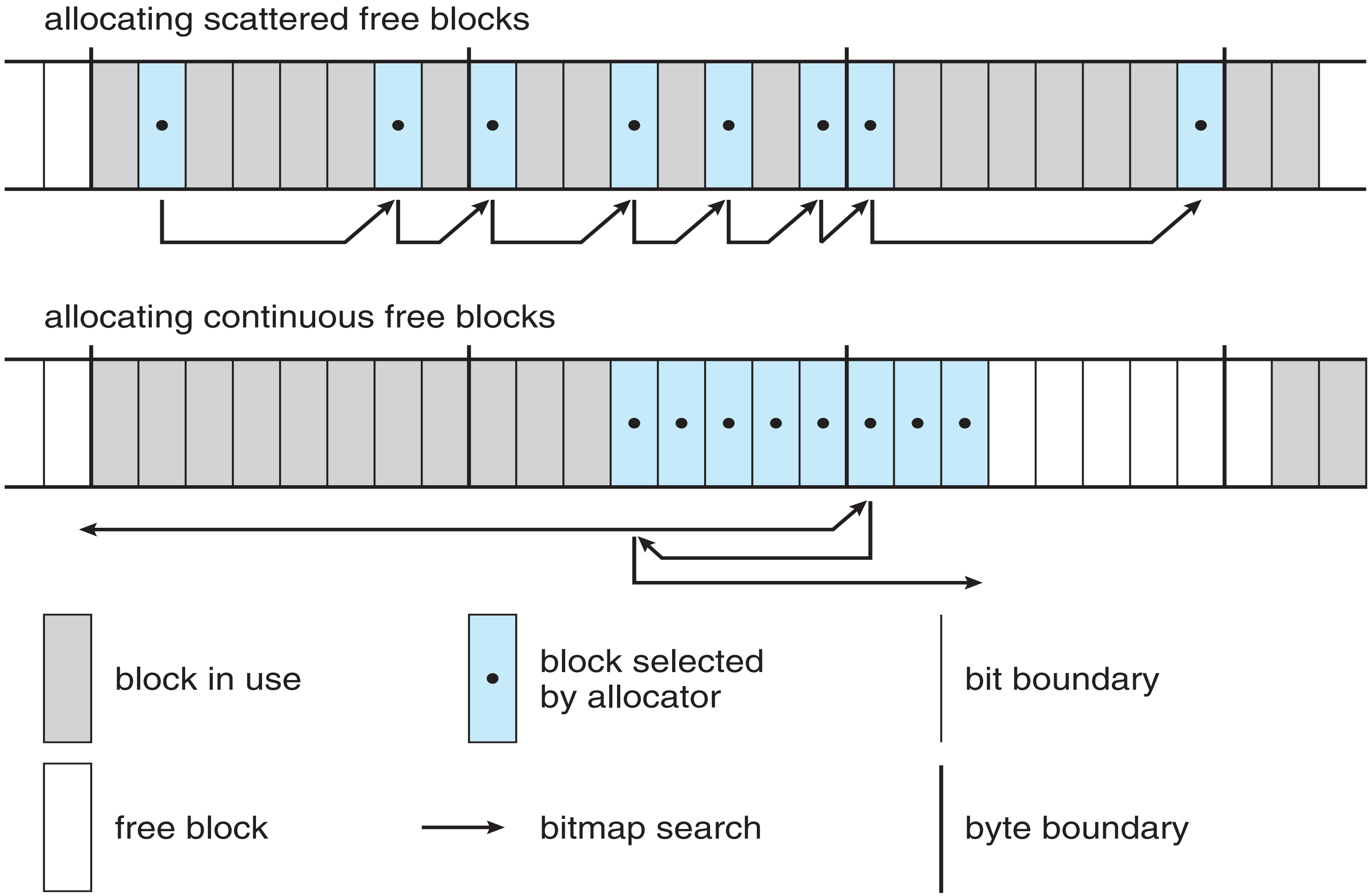
The Linux ext3 File System

- **ext3** is standard on disk file system for Linux
  - Uses a mechanism similar to that of BSD Fast File System (FFS) for locating data blocks belonging to a specific file
  - Supersedes older **extfs**, **ext2** file systems
  - Work on ext4 adding features like extents completed in 2008
    - Extents replace the traditional block mapping scheme used by ext2 and ext3.
    - Now ext4 is the default file system for many Linux distributions including Ubuntu.
  - Of course, many other file system choices with Linux distributions.

The Linux ext3 File System (Cont.)

- The main differences between ext2fs and FFS concern 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
  - ext3 does not use fragments; it performs its allocations in smaller units
    - The default block size on ext3 varies as a function of total size of file system with support for 1, 2, 4 and 8 KB blocks
  - ext3 uses cluster 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 on a **block group**
  - Maintains bit map of free blocks in a block group, searches for free byte to allocate at least 8 blocks at a time

Ext2fs Block-Allocation Policies



Journaling

- ext3 implements **journaling**, with file system updates first written to a log file in the form of **transactions** (i.e. a set of operations that performs a specific task)
  - Once in log file, considered committed
  - Over time, log file transactions replayed over file system to put changes in place
- On system crash, some transactions might be in journal but not yet placed into file system
  - Must be completed once system recovers
  - No other consistency checking is needed after a crash (much faster than older methods)
- Improves write performance on hard disks by turning random I/O into sequential I/O

The Linux Process File System

- The **proc file system** does not store data, rather, its contents are computed on demand according to user file I/O requests
- **proc** must implement a directory structure, and the file contents within; it must then define a unique and persistent inode number for each directory and files it contains
  - It uses this inode number to identify just what operation is required when a user tries to read from a particular file inode or perform a lookup in a particular directory inode
  - When data is read from one of these files, **proc** collects the appropriate information, formats it into text form and places it into the requesting process's read buffer
  - Under Linux, the traditional **ps** command for listing the states of all running processes is implemented as an entirely unprivileged program that simply parses and formats the information from /proc.

THANK YOU

Venkatesh Prasad
Department of Computer Science Engineering
venkateshprasad@pes.edu