CSCI315 – Operating Systems Design

Department of Computer Science
Bucknell University

Linux File System and I/O

Ch 20.7-20.8

This set of notes is based on notes from the textbook authors, as well as L. Felipe Perrone, Joshua Stough, and other instructors.
Xiannong Meng, Fall 2021.

FILE SYSTEM

File Systems

- 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

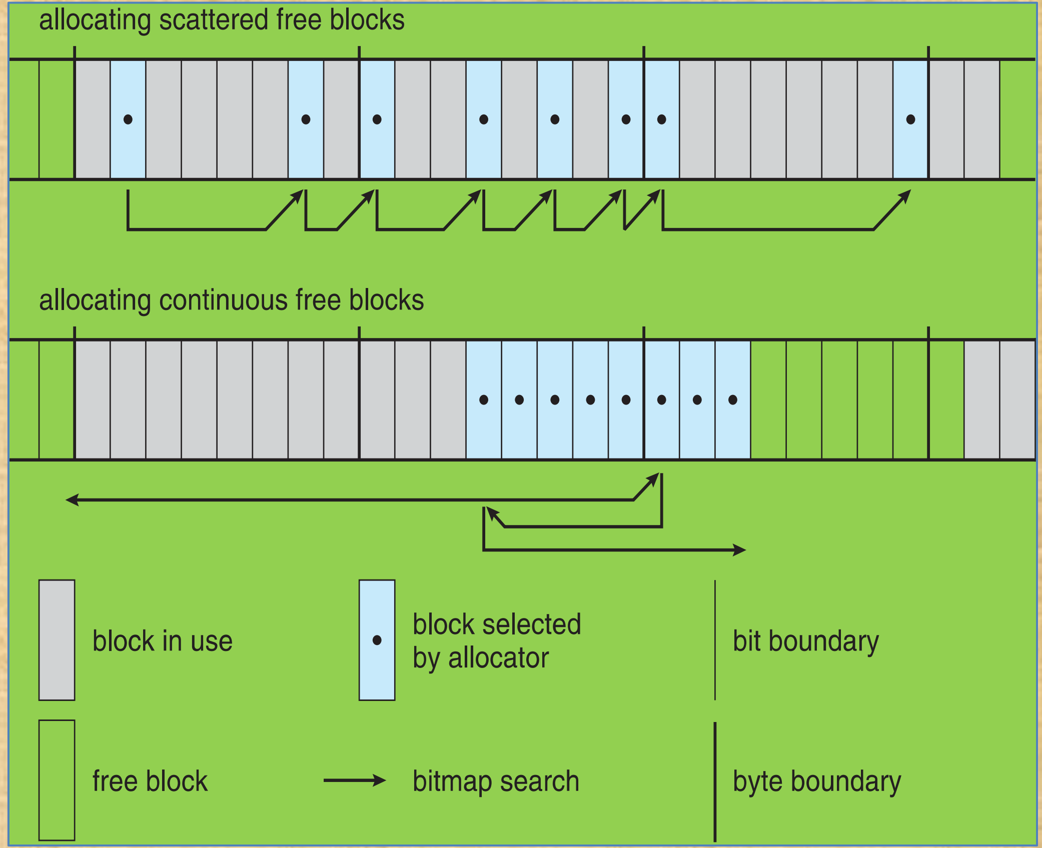
The Linux ext3 File System

- **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 underway on **ext4** adding features like extents
  - Of course, many other file system choices with Linux systems

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



The block numbers of allocated blocks are recorded in their respective **inodes**.

Journaling

- ext3 implements **journaling**, with file system updates first written to a log file in the form of **transactions**
  - 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 Proc 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

The Linux Proc File System

The proc filesystem is a pseudo-filesystem which provides an interface to kernel data structures. It is commonly mounted at /proc. Typically, it is mounted automatically by the system.
-- Linux manual page (**man proc**)

Every process has such a file system that is created when the process is created. The file system is destroyed after the process exits.

The Linux Proc File System: Example

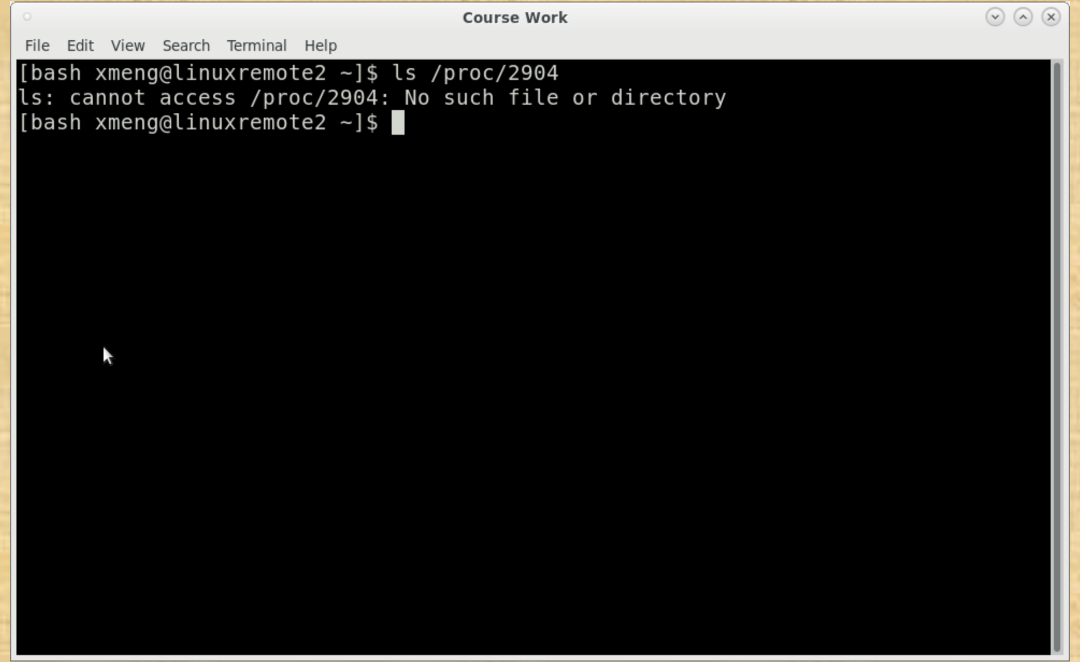
Course Work

File Edit View Search Terminal Help

```
[bash xmeng@linuxremote2 lectures]$ ls /proc/2904/
attr             cwd      map_files   oom_adj        schedstat  task
autogroup        environ  maps        oom_score      sessionid  timers
auxv             exe      mem         oom_score_adj  setgroups  uid_map
cgroup           fd       mountinfo   pagemap        smaps      wchan
clear_refs       fdinfo   mounts      patch_state    stack
cmdline          gid_map  mountstats  personality    stat
comm             io       net         projid_map     statm
coredump_filter  limits   ns          root           status
cpuset           loginuid numa_maps   sched          syscall
[bash xmeng@linuxremote2 lectures]$ ls -l /proc/2904/ | head
total 0
dr-xr-xr-x 2 xmeng cs 0 Nov 17 15:57 attr
-rw-r--r-- 1 xmeng cs 0 Nov 17 15:57 autogroup
-r-------- 1 xmeng cs 0 Nov 17 15:57 auxv
-r--r--r-- 1 xmeng cs 0 Nov 17 15:57 cgroup
--w------- 1 xmeng cs 0 Nov 17 15:57 clear_refs
-r--r--r-- 1 xmeng cs 0 Nov 17 15:45 cmdline
-rw-r--r-- 1 xmeng cs 0 Nov 17 15:57 comm
-rw-r--r-- 1 xmeng cs 0 Nov 17 15:57 coredump_filter
-r--r--r-- 1 xmeng cs 0 Nov 17 15:57 cpuset
[bash xmeng@linuxremote2 lectures]$
```

After process 2904 was created

The Linux Proc File System: Example



After process 2904 completed.

LINUX INPUT AND OUTPUT

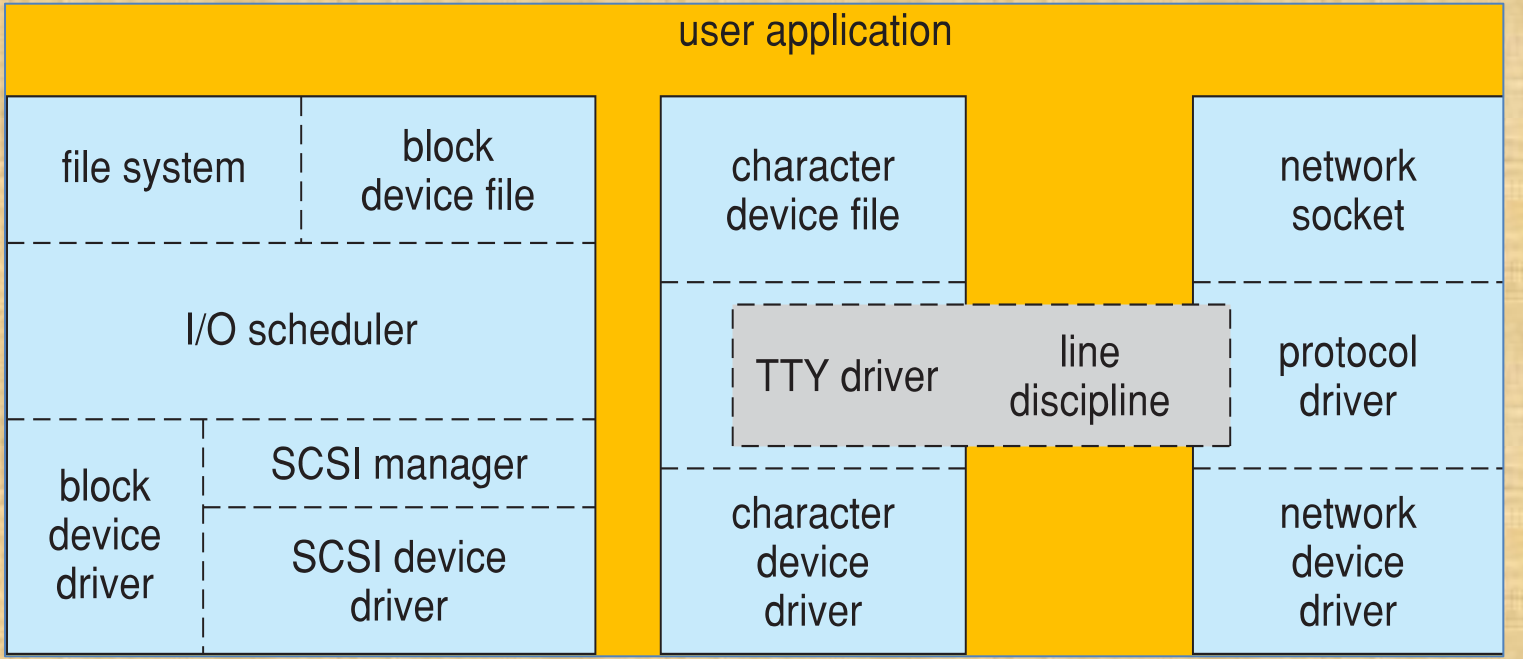
Input and Output

- The Linux device-oriented file system accesses disk storage through two caches:
  - Data is cached in the page cache, which is unified 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 completely 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

Block Devices

- Provide the main interface to all disk devices in a system
- The block buffer cache serves two main purposes:
  - it acts as a pool of buffers for active I/O
  - it serves as a cache for completed I/O
- The **request manager** manages the reading and writing of buffer contents to and from a block device driver
- Kernel 2.6 introduced **Completely Fair Queueing** (**CFQ**)
  - Now the default scheduler
  - Fundamentally different from elevator algorithms
  - Maintains set of lists, one for each process by default
  - Uses C-SCAN algorithm, with round robin between all outstanding I/O from all processes
  - Four blocks from each process put on at once

Device-Driver Block Structure



SCSI: Small Computer Systems Interface, a standard for I/O devices on PCs

Character Devices

- A device driver which does not offer random access to fixed blocks of data
- A character device driver must register a set of functions which implement the driver’s various file I/O operations
- The kernel performs almost no preprocessing of a file read or write request to a character device, but simply passes on the request to the device
- The main exception to this rule is the special subset of character device drivers which implement terminal devices, for which the kernel maintains a standard interface

Character Devices (Cont.)

- **Line discipline** is an interpreter for the information from the terminal device
  - The most common line discipline is **tty** discipline, which glues the terminal's data stream onto standard input and output streams of user's running processes, allowing processes to communicate directly with the user's terminal
  - Several processes may be running simultaneously, **tty** line discipline responsible for attaching and detaching terminal's input and output from various processes connected to it as processes are suspended or awakened by user
  - Other line disciplines also are implemented have nothing to do with I/O to user process – i.e., **PPP** and **SLIP** networking protocols