Exam 2 Notes

I/O Systems and Device Management

Input/Output (I/O) systems are a critical component of any operating system, managing communication between the CPU and peripheral devices such as keyboards, USB drives, and monitors. The operating system interacts with devices through device drivers, which handle the details of device control and communication. Devices are classified as either block or character devices, depending on how they transfer data. Character devices, like keyboards, transfer data one character at a time, while block devices, like USB drives, transfer data in blocks.

Efficient I/O management is crucial for system performance, as it reduces the amount of time the CPU spends waiting for data transfers. I/O operations can be handled using various techniques such as polling, interrupt-driven I/O, or Direct Memory Access (DMA), each offering different performance characteristics based on the frequency and nature of I/O requests. Additionally, device files are identified using major and minor numbers, which allow the kernel to map device files to the correct device drivers.

Device Controller and Device Driver

To start an I/O operation, the operating system uses the device driver to load the appropriate registers within the device controller, which handles communication with the physical device. Each device controller can manage multiple devices and raises interrupts to signal the completion of an I/O operation. Device files are represented by a combination of major and minor numbers, where:

- **Major Number**: Identifies the driver associated with the device.
- **Minor Number**: Identifies the specific device handled by the driver.

I/O Operation Methods

There are several methods for performing I/O operations, each with its own advantages and limitations. Polling, interrupt-driven I/O, and DMA are the most common techniques. Polling involves the CPU constantly checking the status of a device, which can waste CPU cycles. Interrupt-driven I/O, in contrast, allows the device controller to raise an interrupt when the device is ready, freeing the CPU for other tasks in the meantime. Direct Memory Access (DMA) allows devices to transfer data directly to and from memory without involving the CPU, making it the most efficient method for large data transfers.

Interrupt-Driven I/O

Interrupt-driven I/O allows the CPU to perform other tasks while waiting for an I/O operation to complete. When a device is ready, the device controller raises an interrupt by asserting a signal on the interrupt request line, causing the CPU to pause its current task and handle the I/O operation. This method is especially efficient for devices that rarely need to be serviced.

- **Device Controller**: Raises an interrupt when the device is ready.
- **Interrupt Request Line**: The line used to signal the CPU to handle I/O.

Direct Memory Access (DMA)

DMA is used for high-speed data transfer between I/O devices and memory, allowing the CPU to be free during the transfer. The device initiates the transfer, and once the data is copied, the DMA controller sends an interrupt to notify the CPU that the transfer is complete. However, DMA is not automatically invoked for every I/O transfer; it requires specific hardware support and is typically used for large, continuous data transfers.

- **DMA Controller**: Handles data transfer between devices and memory.
- **CPU Involvement**: The CPU is only involved when the transfer completes.

Concurrency and Scheduling

Concurrency refers to the interleaving of processes to simulate parallelism, allowing multiple processes to progress at once. This is crucial in managing multiple I/O operations and ensuring that the system remains responsive. Concurrency allows for better system utilization by keeping the CPU busy while waiting for I/O operations to complete.

Definition of Concurrency

Concurrency is the process of interleaving execution of tasks to make progress simultaneously. In operating systems, concurrency is achieved by switching between processes or threads, allowing the system to handle multiple tasks efficiently, even on a single CPU.

- **Parallelism**: True simultaneous execution of tasks (on multi-core systems).
- **Interleaving**: Simulating parallelism by switching between processes.

Loadable Kernel Modules (LKMs)

Loadable Kernel Modules (LKMs) provide a way to dynamically add or remove functionality from the kernel without needing to reboot the system. LKMs are particularly useful for adding device drivers, file systems, or other kernel features on the fly. This flexibility allows for more efficient memory use and simplifies the development and testing of kernel-level components. One of the key advantages of LKMs is that they can be loaded into or removed from the kernel without requiring a total system recompile or reboot.

Advantages of LKMs

LKMs offer several benefits, including the ability to add or remove features from the kernel dynamically. This modularity allows for more efficient use of system resources and makes it easier to update or test kernel components.

- **Modularity**: Add or remove kernel functionality without rebooting.
- **Efficiency**: Only load necessary modules, reducing memory usage.

Lab Assignment Overview

In this lab, you will implement a loadable kernel module and interact with the I/O system. The lab will cover the process of writing a simple kernel module, compiling it, and loading it into the kernel. You will also explore the interaction between the kernel and devices, particularly how to handle I/O operations using device drivers and DMA.

Lab Objectives

The objectives of the lab are to understand how to write and manage loadable kernel modules and to explore the I/O system. By the end of the lab, you will have experience with creating a kernel module, loading it into the system, and interacting with device drivers for I/O operations.

- **Create a Kernel Module**: Write and load a simple kernel module.
- **I/O Interaction**: Use the module to interact with the I/O system.
- **Device Driver Management**: Manage device files using major and minor numbers.

Step-by-Step Lab Implementation

1. Write a simple kernel module that performs a basic function.
2. Compile the module and load it into the running kernel using `insmod`.
3. Interact with the I/O system by writing a device driver that handles I/O requests.
4. Use major and minor numbers to identify the device files associated with your module.
5. Unload the module from the kernel using `rmmod` and verify that it is correctly removed.

Mass Storage Concepts

Mass storage is a vital component of computer systems, allowing for long-term data storage and retrieval. Devices such as hard disk drives (HDDs) and solid-state drives (SSDs) are commonly used for this purpose. Key factors that affect mass storage performance include seek time, rotational latency, and the effectiveness of disk scheduling algorithms. RAID configurations are also essential for ensuring data reliability and improving storage performance.

Understanding how data is managed on physical storage devices is crucial for optimizing both performance and reliability in modern computer systems. Disk scheduling algorithms help minimize wait times by controlling the order in which data is accessed, while RAID configurations introduce redundancy to protect against data loss.

Seek Time and Rotational Latency

Seek time refers to the time it takes for the read/write head of a hard drive to move to the correct track on the disk. This process is a major factor in overall disk access time. Rotational latency, on the other hand, is the time taken for the desired disk sector to rotate under the read/write head. Both factors significantly affect the performance of mechanical hard drives.

- **Seek Time**: The time required for the read/write head to reach the correct track.
- **Rotational Latency**: The time needed for the sector to rotate into position under the read/write head.

Disk Scheduling Algorithms

Disk scheduling algorithms optimize the order in which disk requests are handled, reducing seek time and improving overall performance. Without proper scheduling, disk heads may move inefficiently across the disk surface, resulting in longer access times. Algorithms like First-Come, First-Served (FCFS), Shortest Seek Time First (SSTF), and SCAN offer different approaches to managing I/O requests.

Disk Scheduling Techniques

- **FCFS**: Handles requests in the order they arrive, but can lead to inefficient head movements.
- **SSTF**: Prioritizes the request closest to the current head position, minimizing seek time.
- **SCAN**: Moves the read/write head in one direction, servicing requests along the way, then reverses direction.

RAID Arrays

RAID (Redundant Arrays of Independent Disks) enhances data reliability and performance by distributing data across multiple disks. RAID configurations such as RAID 1 (mirroring) and RAID 5 (distributed parity) ensure that data can be recovered in case of drive failure, while also improving read and write speeds through parallel access to disks.

RAID Levels

- **RAID 1**: Uses mirroring to duplicate data across two or more disks, providing full redundancy.
- **RAID 5**: Distributes both data and parity across multiple disks, offering a balance between redundancy and storage efficiency.

Disk Scheduling Performance Example

When comparing disk scheduling algorithms in practice, the differences in efficiency become clear. For example, consider a disk head initially positioned at cylinder 250, with requests at cylinders 260, 350, 100, 10, and 410. Depending on the scheduling algorithm, the total movement of the disk head can vary:

Scheduling Algorithm Comparison

- **FCFS**: Traverses 1240 cylinders to complete all requests.
- **SSTF**: Optimizes head movement to only 560 cylinders by choosing the closest request.
- **SCAN**: Traverses 560 cylinders by sweeping in one direction before reversing.
- **C-SCAN**: Scans in one direction and resets, traversing 350 cylinders actively.

Key Concepts Summary

Summary of Key Concepts

- **Seek Time and Rotational Latency**: Major factors in disk performance, particularly in mechanical drives.
- **Disk Scheduling**: Optimizes data access by controlling the order in which disk requests are serviced.
- **RAID**: Provides redundancy and performance benefits through data distribution across multiple drives.
- **Algorithm Efficiency**: Different scheduling algorithms lead to varied head movement and access times.

Mass storage systems are essential for both performance and data reliability. Disk scheduling algorithms and RAID configurations help optimize storage efficiency, reduce access times, and protect against data loss.

File System Organization

A file system is essential for organizing and managing data on storage devices. Humans organize files using a directory structure, which helps in locating and managing files. The directory structure provides a logical organization that abstracts the underlying hardware details.

File systems typically consist of files and directories, with each file containing its data and metadata. Metadata includes information like file size, file type, access permissions, and timestamps, which are critical for managing file operations efficiently.

File Operations and Metadata

- **File Operations**: Include actions like opening, reading, writing, and seeking within files.
- **Metadata**: Includes file size, type, access rights, and time created/modified.

File Access Methods

Files can be accessed sequentially or directly. Sequential access reads data from the beginning of the file to the end in order, while direct access allows reading or writing at any location within the file. Direct access requires a seek operation to move the file pointer to the desired position.

Sequential vs. Direct Access

- **Sequential Access**: Reads data in order from start to end.
- **Direct Access**: Allows reading/writing at any location in the file using `seek()`.

File Sharing and Access Control

File sharing among processes and users is controlled through access permissions. File access control defines the rights for each user, such as read, write, or execute permissions. Additionally, the kernel keeps track of files opened by a process.

File Access Control

- **Access Rights**: Define how users can interact with a file (read, write, execute).
- **Kernel Management**: The kernel maintains information about open files per process.

Symbolic Links and File Mounting

Symbolic links are shortcuts that point to other files or directories, and they can create loops, which the system must manage. When mounting, files and directories from different devices (e.g., DISK2) can logically appear as part of another device's file system (e.g., DISK1).

Symbolic Links and Mounting

- **Symbolic Links**: Can create loops, but are distinguishable from regular files.
- **File Mounting**: Files from different devices can appear unified in a single file system.

Virtual File Systems (VFS)

A Virtual File System (VFS) provides an abstraction layer over different file systems, translating file operations from the OS to the mounted file systems. VFS allows the system to support various file system types seamlessly.

Virtual File System (VFS)

- **VFS Role**: Abstracts different file systems and translates file operations to each.
- **System Flexibility**: Allows different file systems to coexist on the same machine.

File Allocation Methods

Contiguous file allocation requires files to be stored in consecutive blocks, which causes external fragmentation and requires knowing the file size beforehand. Linked-list allocation solves this, but suffers from high overhead for random access.

File Allocation Disadvantages

- **Contiguous Allocation**: Causes external fragmentation and requires prior knowledge of file size.
- **Linked-List Allocation**: Has high overhead for random access due to traversal.

Inode and Indexed Allocation

In UNIX-based file systems, the inode structure uses direct, indirect, double indirect, and triple indirect blocks for block allocation. FAT (File Allocation Table) is another indexed allocation method, but both FAT and linked-list allocations suffer from traversal latency.

Inode Block Allocation

- **Direct Blocks**: Store data directly in blocks.
- **Indirect Blocks**: Use pointers to data blocks, supporting larger files.

Free Space and File System Optimization

Free space management keeps track of unallocated blocks within the file system. File caching is a common method to enhance file system performance by reducing the number of accesses to slower storage.

Free Space and Optimization

- **Free Space Management**: Tracks unallocated blocks in the file system.
- **File Caching**: Improves performance by storing frequently accessed data in faster memory.

File System Consistency

To maintain file system consistency, log-based transaction file systems are used to track changes, ensuring that if the system crashes, the file system can recover to a consistent state.

Consistency Maintenance

- **Log-Based Transaction Systems**: Ensure file system integrity and recovery after crashes.

Network File Systems (NFS)

NFS (Network File System) is both a specification and an implementation of a protocol that allows files to be accessed over a network, making remote file systems appear as if they are local.

NFS Overview

- **NFS**: Allows remote file systems to be mounted and accessed over a network.