SNS COLLEGE OF TECHNOLOGY

Coimbatore-35
An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A+' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF INFORMATION TECHNOLOGY

23CST202 – Operating Systems

II YEAR - IV SEM

UNIT 4 – FILE SYSTEMS

Syllabus

- UNIT I OVERVIEW AND PROCESS MANAGEMENT 9
- Introduction - Computer System Organization, Architecture, Operation, Process Management – Memory Management – Storage Management – Operating System – Process concept – Process scheduling – Operations on processes – Cooperating processes – Inter process communication. Threads - Multi-threading Models – Threading issues.
- UNIT II PROCESS SCHEDULING AND SYNCHRONIZATION 10
- CPU Scheduling - Scheduling criteria – Scheduling algorithms – Multiple-processor scheduling – Real time scheduling – Algorithm Evaluation. Process Synchronization - The critical-section problem – Synchronization hardware – Semaphores – Classical problems of synchronization. Deadlock - System model – Deadlock characterization – Methods for handling deadlocks – Deadlock prevention – Deadlock avoidance – Deadlock detection – Recovery from deadlock.
- UNIT III MEMORY MANAGEMENT 9
- Memory Management - Background – Swapping – Contiguous memory allocation – Paging – Segmentation – Segmentation with paging. Virtual Memory - Background – Demand paging – Process creation – Page replacement – Allocation of frames – Thrashing.
- UNIT IV FILE SYSTEMS 8
- File concept - Access methods – Directory structure – Files System Mounting – File Sharing – Protection. File System Implementation - Directory implementation – Allocation methods – Free-space management.
- UNIT V I/O SYSTEMS 9
- I/O Systems - I/O Hardware – Application I/O interface – Kernel I/O subsystem – Streams – Performance. Mass-Storage Structure: Disk scheduling – Disk management – Swap-space management – RAID – Disk attachment – Stable storage – Tertiary storage. Case study: Implementation of Distributed File system in Cloud OS / Mobile OS.

- L :45 P:0 T: 45 PERIODS

FILE SYSTEMS

- File concept
- Access methods
- Directory structure
- Files System Mounting
- File Sharing
- Protection

File-System Implementation

- In this section, we learn different data structures that used to assist in the implementation of file systems.
- There are several on-disk and in-memory structures used for the implementation of file systems.
- The on-disk structures are kept in the disks.
- The on-disk structures contain information about how to boot an OS stored in the disk, total number of disk blocks, number and location of free disk blocks, directory structure and individual files.
- The in-memory structures are kept in the main memory.
- The in-memory structures are helpful for file-system management, caching and so on.

File-System Implementation

On-Disk Structures

- In this section, we understand the functionalities of different on-disk data structures.

File-System Implementation

Boot control block

- Operating system is kept in the boot control block.
- If the disk has no operating system, this block is empty.
- In UNIX, the boot control block is called as a boot block.
- In NTFS, the boot control block is called as a partition boot sector.
- The boot control block is usually the first block of the volume where the file system is kept.
- If a partition of a disk has an operating system, information about how to boot.

File-System Implementation

Volume control block

- This data structure contains details about a volume (partition).
- The information maintained in the volume control block are number of blocks in the partition, size of each block, number of free blocks in the partition, free-block pointers (addresses of free disk blocks) and so on.
- In UNIX the volume control block is called a superblock and is the block next to the boot block.
- In NTFS, these details are stored in the master file table.

File-System Implementation

Directory structure (for each file system)

- The directory structure is used to organize files.
- In the directory structure, the names of files and associated information are kept.
- In UNIX, the directory structure includes file names and associated inode numbers.
- In NTFS, these details are stored in the master file table.

File-System Implementation

Per-file file control block (FCB)

- For each and every file, information about that file is maintained in a file control block.
- The FCB has a unique identifier number to allow association with a directory entry.
- In UNIX, the per-file file control block is nothing but the inode.
- The inode has an inode number, which is the unique identifier.
- In NTFS, the details about the file is stored in the master file table.

File-System Implementation

In-Memory Structures

These are data structures that are maintained inside the main memory.

- Mount table

– Information about each mounted volume is maintained in the mount table.

- Directory-structure cache

– Holds directory information of recently accessed directories.

– If the same directory has to be accessed again, it is not necessary to read from the disk.

– The details can be taken from the directory-structure cache.

File-System Implementation

- System-wide open-file table

– This data structure is common to all the processes present in the system. This contains a copy of the FCB of each open file.

- Per-process open-file table

– This is a table that is available for each and every process. This per-process open-file table points to the appropriate entry in the system-wide open-file table.

- Buffers

– These are buffers that are kept in the memory to hold the contents of disk blocks.

File-System Implementation

- Each buffer can hold the contents of one disk block.
- When contents of a disk block are read from the disk, they are stored in these buffers kept in memory.
- If the contents of the same disk block are needed again, the contents are taken from the buffer and need not be read from the disk.
- Similarly, the contents present in the buffer may be modified and need not be written to the disk for each and every modification.
- It is enough to write the contents of the buffer to the disk when the buffer is to be used for some other disk block contents.

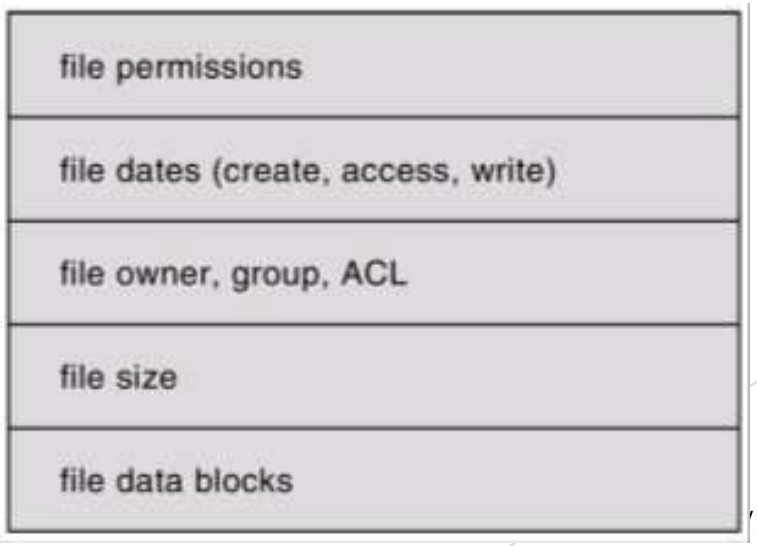
File Operations

Create a new file

- The application program calls the logical file system and gives the name of the file to be created to the logical file system.
- The logical file system knows the name of the directory structures.
- It finds the name of the directory in which the file is to be created from the file name given by the application program.
- The logical file system allocates a new FCB.

File Operations

- In the case of UNIX, a new inode is allocated.
- The system reads the appropriate parent directory into memory.
- It updates the directory with the new file name and FCB and writes the directory back to disk.
- Figure shows a typical file control block.
- The FCB has information about the file like the owner of the file, file size, file permissions and so on.



File Operations

- **Open an Existing File**

- The open() call called by the application program passes a file name to the logical file system.
- The call first searches the system-wide open-file table to see if the file is already in use by another process.
- If it is, a per-process open-file table entry is created pointing to the existing system-wide open-file table entry.
- If file is not already open, the directory structure is searched for the given file name.
- There is a possibility that parts of the directory structure are cached in memory.

File Operations

- If the directory structure is present in the cache, it is taken from the cache.
- Else, the directory structure is read from the disk.

- Once the file is found, the FCB is copied into an entry in the system-wide open-file table in memory.
- The FCB entry also keeps track of the number of processes that have opened the file.
- An entry is made in the per-process open-file table.
- This entry points to the system-wide open-file table.

File Operations

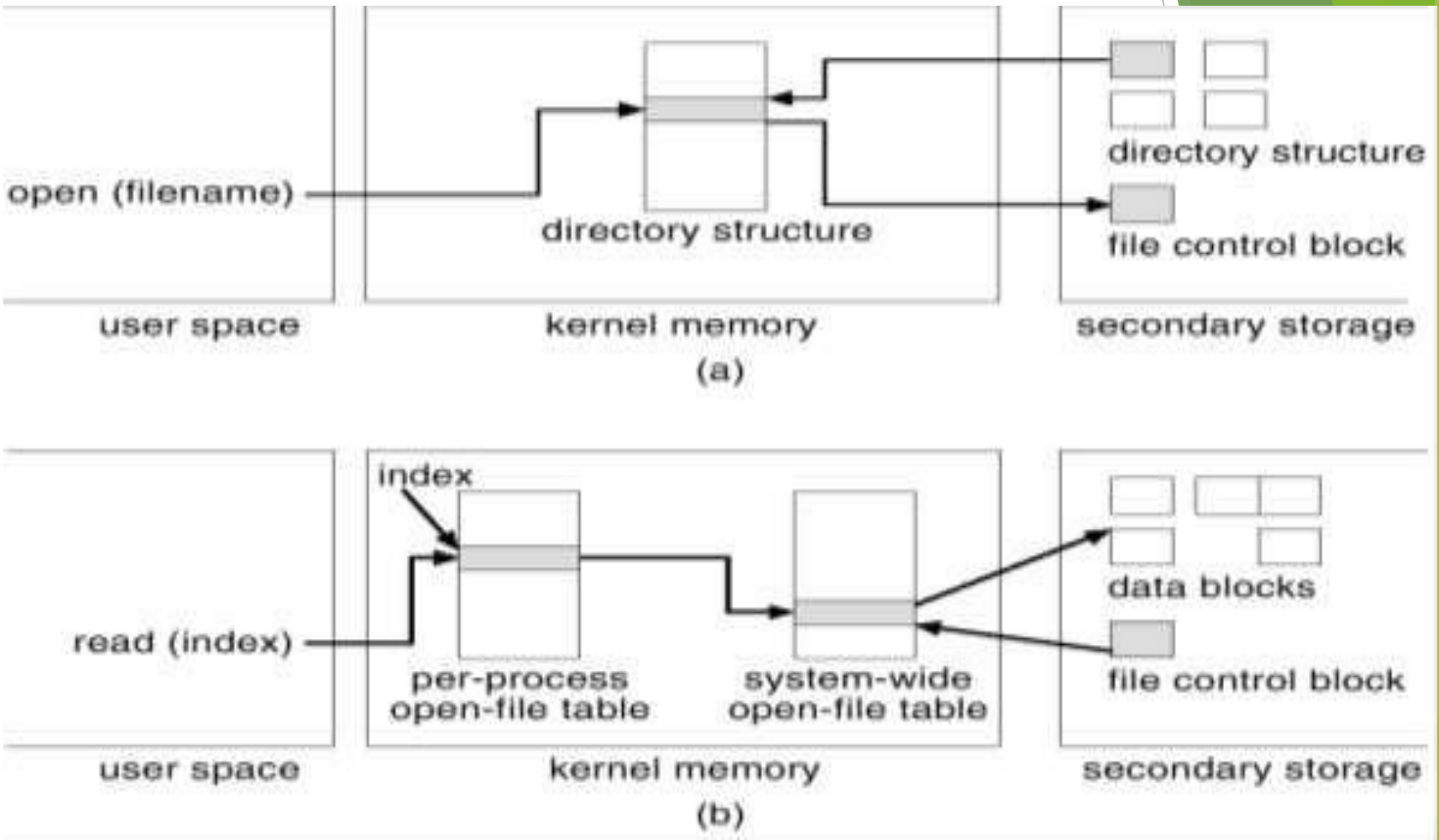
- The FCB entry also has information about where the next read/write should be done on the file (file offset), access mode in which the file is open.
- open() returns a pointer to the entry in the per-process file-system table. All file operations after the open() system call use this pointer.
- In UNIX this pointer is called the file descriptor (file handle in Windows).

File Operations

- **Close a File**

- When a process closes a file, the per-process open-file table's entry is removed.
- The count (count of the number of processes using the file) in the system-wide open-file table entry is decremented.
- When the count becomes zero, the updated metadata is copied to the directory structure in the disk and the system-wide open-file table entry is removed.

File Operations



File Operations

- Figure shows that the open call accesses the directory structure in memory.
- If the directory structure is not cached in memory, it is read from the disk.
- The file control block is accessed using the directory structure.
- If a copy of the FCB is not present in the memory, it is read from the disk.

File Operations

- Figure shows how the read system call uses the in-memory data structures.
- The read uses the index returned by the open call to access the entry in the per-process open-file table.
- The entry in the system-wide open-file table is obtained using the pointer from the per-process open-file table.
- The file control block is accessed from the system-wide open-file table and the data blocks are accessed using the entries in the FCB.

Directory Implementation

- This section explains two different ways in which a directory can be implemented.
- In the first method, a linear list of file names is maintained with pointers to the data blocks.
- This method is simple to program but time-consuming to execute.
- To create a new file, the directory is searched to find if another file with a similar name exists.
- If no such file exists, a new entry is added to the end of the directory.

Directory Implementation

- To delete a file, the directory is searched for that file and the space allocated to it is released.
- To reuse this deleted entry, the entry is marked as unused by assigning it a special name or it is attached to a list of free directory entries or the last entry in the directory is copied to the freed location and the length of the directory is decreased.
- The entries can also be maintained as a linked list to reduce the time required for deletion.
- The disadvantage of a linear list is that finding a file requires linear search and this makes access slow.

Directory Implementation

To reduce this search time

- Operating systems implement a software cache to store the most recently used directory information.
- Maintaining a sorted list also allows a binary search and reduces the search time.

Directory Implementation

- But maintaining a sorted list is difficult.
- When new entries are added, the new entries should be added to the appropriate position.

- Using a hash data structure
- –name and returns a pointer to the file name in the linear list.
- This reduces the
- – In addition to having a linear list for storing the directory entries, a hash data structure can also be used.
- The hash table takes a value computed from the file search time.
- But provision must be made for collisions, that is, it is to be ensured that two file names do not hash to the same location.