

## Review

### Deadlocks

- Non-sharable resources
- Necessary conditions for a deadlock to occur
  1. Mutual exclusion
  2. Hold and wait
  3. No preemption
  4. Circular wait
- Resource graph
- Use resource graph to detect deadlock:  
Check if there is any circle in the graph
- Methods for handling deadlocks
  1. The Ostrich algorithm
  2. Detection and recovery
    - Detect cycles in resource graph
    - Depth-first search
    - Restore modified files for recovery

## Review

### Deadlocks (Cont.)

- **Methods for handling deadlocks**

#### **3. Deadlock prevention**

**Negating one of four conditions**

- (a) Mutual exclusion:**  
spooling
- (b) Hold and wait:**  
request all resources in advance
- (c) No preemption:**  
take resource away
- (d) Circular wait:**  
numerically order resources

#### **4. Deadlock avoidance**

- **Banker's algorithm for single resource**
- **Resource trajectories**
- **Banker's algorithm for multiple resources**

## File systems

- **The best way to store information:**
  - Store all information in virtual memory address space
  - Use ordinary memory read/write to access information
  - Not feasible: no enough address space
  - When process exits, information is lost
  
- **Another approach:**
  - Define named objects called files which hold programs and data
  - Files are not part of the address space
  - O.S. provides special operations (system calls) to create/destroy files
  - Files are permanent records

## User's view of the file system

- A file system consists of two distinct parts
  1. A collection of files, each storing related data
  2. A directory structure, which organizes and provides information about all the files in the system
  
- Three common file organizations
  1. Unstructured byte sequence

The basic unit for reading and writing is byte, sequential access.

Example: Unix.

## 2. Sequence of fixed-size records

The basic unit is a record, sequential access. Records can be read and written, but cannot be inserted or deleted in the middle of a file.

Example: CP/M.

## 3. A tree of disk blocks

The basic unit is a record. Each block holding  $n$  keyed records. Records can be looked up by key and inserted. Blocks can be split.

Used on mainframe computers.

- **File types**
  - **Regular files:** contain data bytes
  - **Directories:** contain the names of other files
  - **Character special files:** e.g. terminals, printers
  - **Block special files:** e.g. disks
  
- **File access methods**
  - **Sequential access:** start at the beginning, read the bytes or records in order
  - **Random access:** read bytes or records in any order
  
- **File attributes:**  
**Extra information associated with each file**

- **File operations**

- **Create**
- **Delete**
- **Open: bring attributes and disk addresses to memory for fast access**
- **Close: put back to disk**
- **Read**
- **Write**
- **Append**
- **Seek: put the pointer to a specific location in the file**

- **Directories**

1. **Single directory shared by all users**

Different users cannot use the same file name.

2. **One directory per user**

Different users can use the file name. But each user can keep only a sequence of files, no any group support.

3. **Arbitrary tree per user**

Flexible. Used in Unix.

## File system implementations (designer's view)

- How to store  $n$  bytes on disk
  - Allocate  $n$  consecutive bytes of disk space

For insertion, have to move the file (slow)

- The file is split up into a number of (not necessarily contiguous) fixed size blocks

More expensive to keep track of where the blocks are.

- **Block size**

- **Candidates: sector, track, cylinder**
- **The time to read a block from disk:**

**Seek time + rotation time + transfer time**

**Example:**

**A disk with 32768 bytes per track**

**Rotation time: 16.67 msec**

**Average seek time 30 msec**

**Block size  $K$  bytes**

**The time to read a block:**

$$30 + 16.67/2 + (K/32768) \times 16.67$$

- **Large block size: poor disk space utilization**
- **Small block size: reading a block is slow**
- **Compromise between the speed and utilization**
- **General block size: 512, 1K or 2K**

- **Keep track of free blocks**

- **Linked list**

Use several blocks to store free block numbers. Link these blocks together. Search is fast.

Example: 1K block size. 16-bit disk block number (20 M disk). Need 40 blocks to hold all 20K block numbers.

- **Bit map**

$n$  blocks require  $n$  bits. Smaller space. Search is slow. May be put in memory.

Example: 20 M disk requires only 20K bits (3 blocks).

- **Keep track of the blocks of each file**

- Store blocks consecutively

When file grows, may cause problem.

- Linked list

Use the last two bytes as a pointer to point to the next block in the file.

**Problems:**

1. The number of bytes in a block is no longer power of 2
2. Random access is slow

– File allocation table (FAT)

- \* A table has one entry for each disk block.
- \* All blocks of a file linked together.
- \* The directory keeps the first block number of each file.
- \* The table is stored in memory.
- \* Example system: MS-DOS
- \* Problem:

Pointers for all files on the disk are mixed up in the same table. An entire FAT is potentially needed for accessing one file.

- **Keep the block lists for different files in different places**

**I-node (index node) in Unix.**

- **Each file has a small table called I-node**
- **I-node contains attributes, first 10 blocks of the file, and 3 indirect block numbers.**
- **Single indirect block: points to data blocks**
- **Double indirect block: points to single indirect blocks**
- **Triple indirect block: points to double indirect blocks**

– **Example:**

**For 1K disk block and 32-bit disk address, how large a file could be?**

- \* **Single indirect block holds 256 disk addresses**
  
- \* **Double indirect block holds 256 single indirect blocks**
  
- \* **File size < 10 blocks: no indirect blocks**
  
- \* **File size between 10 and 266 blocks: single indirect**
  
- \* **File size between 267 and  $266 + 256^2 = 65,802$  K blocks, use double indirect**
  
- \* **File size between 64 M and 16 G, triple indirect**
  
- \* **File size > 16 G, cannot handle**

- **Advantage: fast random access**

At most three disk references are needed to locate the disk address of any byte in the file. When file is open, its I-node is brought into memory until the file is closed.

- **Example of finding the 300th block in a file:**

$$300 - 266 = 34$$

Follow double indirect to the first single indirect, the 34th block.

No need to follow the 299 blocks before this block as in the linked list case.

## Directory organization

- **Single directory in the system.**  
Example: CP/M
- **Hierarchical directory tree.**

– **Example 1: MS-DOS**

**Attributes: 8 1-bit flags:  $b_7, b_6, \dots, b_0$**

$b_0 = 1$ : **read only file**

$b_1 = 1$ : **hidden file**

$b_2 = 1$ : **system file**

$b_4 = 1$ : **a subdirectory**

– **Example 2: Unix**

## Shared files

- A shared file appears simultaneously in different directories belonging to different users
- File system is a directed acyclic graph, not a tree
- Problems:

When directories contain disk addresses (e.g. CP/M), cannot share file.

- Solution:
  - Directory does not point to disk addresses. Point to a data structure associated with the file.
  - Use Link Count to record how many users are sharing the file.
  - When performing “remove file”, only when count = 0, remove the file, otherwise reduce the Link Count.

- **Example: Unix.** To share a file, different directories point to the same i-node.
- **Problem with pointing to the same data structure**

After file owner removes the file, counting information may still be charged to file owner.

- **Solution: use symbolic linking:**

Create a new file, of type LINK. Enter that file in another user's directory. The new file only contains the path to the shared file. Can be used on different machines.

- **Problem with symbolic linking:**

Extra overhead to access file: read extra block to get the i-node.

- **Unix provides both ways**

- **ln file1 file 2**  
Points to the same i-node. Can be seen by  
ls -i file1  
ls -i file2
  
- **ln -s file link**  
Create a symbolic link: link  
ls -l will show  
l - - - - - link@→ file  
more link is equivalent to more file

## Review

### File systems

- **User's view of the file system**
  - **Three file organizations**
  - **Four types of files**
  - **File access methods**
  - **File attributes**
  - **File operations**
  - **Directories**
  
- **File system implementations**
  - **Store  $n$  bytes on disk**
  - **The time to read a block from disk**  
**Seek time + rotation time + transfer time**
  - **Compromise between the speed and utilization**
  - **General block size**
  - **Keep track of free blocks**
    - \* **Linked list**
    - \* **Bit map**

## Review

### File system implementation (Cont.)

- Keep track of the blocks of each file
  - Sequential, file change may be a problem
  - Linked list, slow random access
  - File allocation table (FAT), all files share one table
  - I-node, a small structure per file
- Directory organization
- Looking up a file in Unix: get directory and I-node alternatively
- Shared files
  - Point to the same I-node
  - Symbolic linking

## File system consistency

- After system crashes, the file system may be in an inconsistent state.
  
- A utility program checks:
  1. Block consistency: a block is either free or in a file.
    - Two counters per block: one counter counts the times the block presents in a file; another counter counts the times the block presents in the free list.
    - Read all I-nodes:  
for each block number in I-node, increment the first counter
    - Check the free list, increment the second counter

- **Consistent:** counter1/counter2 = 1/0 or 0/1
  - **Missing block:** 0/0  
Correction: put the block in free list
  - **Duplicated block in free list:** 0/2  
Correction: delete one from free list
  - **Duplicated data block:** 2/0  
Correction: allocate a free block, copy the data of the block into it and insert the copy to one of the file
  - 1/1: should be corrected to 1/0
2. **File consistency:** check how many users are sharing a file
- **One counter per file (per I-node)**
  - **Starts from root directory, inspect each directory. For every file, increment the counter for the file's I-node**

- Compare the counters with the Link Count in the I-node
  - \* Consistent: Link Count = counter
  - \* Inconsistent: Link Count  $\neq$  counter
  - \* Link Count  $>$  counter: all file removed, I-node still not removed, waste space
  - \* Link Count  $<$  counter: file may point to a released I-node
- Solution: Force the Link Count = actual number of directory entries

### 3. Other heuristic checks

- I-node number  $>$  it should be
- Strange mode
- Directories with too many entries

## File system performance

- Disk access is about 100,000 times slower than memory access
  
- Reduce the number of disk access needed
  
- Block cache: keep some blocks (logically belong to the disk) in memory
  - Read a block, first check cache
  
  - If the block is not in cache, read it into cache, then use it.
  
  - For full cache, replace one block
  
  - Replacement algorithm: similar to virtual memory (FIFO, LRU,...), but need to consider special features:

- \* **Blocks may not have good locality**
- \* **Which blocks will be needed again soon**

Indirect blocks are seldom used again. Partially full data block may be needed soon (being written).

- \* **Which blocks are essential to the consistency of the file system**

I-node, indirect, and directory blocks are essential, and need to be written to disk as soon as modified.

Data blocks should not stay in cache for too long without writing them out (possible data loss when system break down).

\* Two ways to deal with this:

(1) Provide a system call which forces all modified blocks out into disk. Unix system call SYNC, called every 30 sec.

(2) Write through cache. All modified blocks are written back to disk immediately.

Example: MS-DOS.

- Reduce the amount of disk arm motion.
  - Allocate the blocks that are likely to be accessed in sequence in the same cylinder.

- How to get consecutive free blocks.

If linked list is used for free blocks, keep track of disk usage not in blocks, but in groups of consecutive blocks (e.g. in units of 2 blocks. Allocation unit is still a block, but 2 consecutive blocks in a file are consecutive on disk).

- Reduce rotation time:

Place consecutive blocks in a file in the same cylinder, but interleaved for maximum throughput.

Example:

Rotation time = 16.67 ms

Block transfer time 4 ms

4 way interleave. One rotation can get 4 blocks.

- For the system using I-node

First access I-node then data block. Put I-node close to data block.

- Usually I-node placed at the start of the disk. On the average, data block is  $1/2$  of the number of cylinders away.
- Put I-node at middle of the disk. Average distance:  $1/4$  of the number of the cylinders.
- Divide disk into cylinder groups. Each with its own I-nodes, blocks and free list. For an I-node in each group. Try to allocate blocks within the group.

## Security

- Some common problems
  - Abuse of valid privileges  
e.g. on Unix, a super user can do anything.
  - Trojan Horse  
Modify a normal program to do nasty things in addition to its normal function.  
e.g. leave a program lying around that looks like the login process when people type passwords, remember them.
  - Spoiler  
Use up all resources and make system crash.  
e.g. grab all disk space or create thousands of processes.
  - Worm or virus  
A Trojan Horse that is also capable of spreading itself from machine to machine.

- Famous historical security flaws

- Unix utility `lpr`  
Print a file with an option to remove the file after printing.  
May delete password file this way.
- TENEX O.S. (Used on DEC-10 computers)
  - \* Can call a user function on each page fault
  - \* To access a file, a program had to present a password.
  - \* Can find a password by putting password crossing the page boundary. For password length  $n$  and 128 characters, at most  $128n$  times are needed to determine the password instead of  $128^n$  times.

– **Sendmail/finger worm (Internet worm)**

- \* **Disabled thousands of computers in Nov. 1988.**
- \* **Used two bugs in Berkeley Unix system.**
- \* **Sendmail attack:**  
Worm can mail a copy of program, get it executed and set up a Trojan Horse on machine.
- \* **Finger attack:**  
Give a carefully designed long name to finger which overflow buffer, modifies stack, causing /bin/sh to be executed.

## **Review**

- **File system consistency**
  - Utility program checks
    - (1) Block consistency
    - (2) File consistency
  
- **File system performance**
  - Reduce disk access (block cache)
  - Reduce disk arm motion
  - Reduce rotation time
  
- **Security**
  - Common problems
  - Historical security flaws

- **How to test a system's security**

Make sure the system can withstand the following attacks

- Request memory pages, disk space or tapes and just read them (to see if the system erases the information before allocating it).
- Try illegal system calls, or legal system calls with illegal parameters to confuse the system.
- Start login in and then hit DEL, or BREAK halfway through the login sequence. May kill password checking program and login successfully without a password.
- Try modifying complex operating system structures (security related) in user space.
- Look for manual that says “Do not do X.” Try as many variations of X as possible.
- Fool the user by writing a program that types “login:” on the screen and go away (record password).

- **Password:**

a secret piece of information used to establish the identity of a user.

- Password file: a series of ASCII lines (encrypted password), one line per user.
- When login, user types password. O.S. encrypts the password and compare it with that in password file.
- In theory, for 7 char password length, randomly chosen from 95 printable ASCII char, there will be  $95^7 \approx 7 \times 10^{13}$  possibilities.

At 1000 encryptions per second, requires 2000 years to build the list to check the password file.

- In practice, people use first names, last names, street names, city names, some common words, car license plate numbers,...

Research shows 86% matched.

- **Solution:** Concatenate an  $n$ -bit random number with each password. Encrypted together and stored in password file. For one password, has to encrypt  $2^n$  strings to match the password file. Unix uses  $n = 12$ .
- This way can only prevent someone guessing your password off-line, but cannot prevent someone trying to login into your account on-line.
- Choose password as random as possible.

## Protection

- The objects need to be protected:  
CPU, memory segments, terminals, files, semaphores,...
- Each object has a name and a set of operations.  
Example:  
file: name, read/write  
semaphore: name, up/down
- Protection domain: a collection of (object, right) pairs.  
Right: permission to perform one of the operations
- At any time, each process runs in some protection domain.

- **Keep track of which objects belongs to which domain**

- 1. Protection matrix:**

Rows are the domains and the columns are the objects. Each matrix entry lists the rights.

Problem: matrix is large and sparse.

- 2. Access control lists: store the matrix by columns.**

Associated with each file. Indicate which users are allowed to perform which operations.

- **General form: each file has a list of (user, privilege) pairs.**

**\* Example:**

Four users A, B, C, and D, belong to groups: system, staff, and student.

File0: (A, \*, RWX)

File1: (A, system, RWX)

File2: (A, \*, RW-), (B, staff, R-), (D, \*, R-)

File3: (\*, student, R-)

File 4: (C, \*, —), (\*, student, R-)

File0 can be RWX by any process with uid=A, gid=any.

File1 can be RWX by uid=A, gid=system.

File2 can be RW by uid=A, gid=any.

File2 can be R by uid=B, gid=staff.

File2 can be R by uid=D, gid=any.

File3 can be R by uid=any, gid=student.

Processes with uid=C, gid=any have no access to file4, but all other processes with uid=any, gid=student can read file4.

- **Compressed form:** users are grouped into classes.  
e.g. in Unix, each file has 9 bits RWXR-WXRWX for self, group and other three classes.
- **Default:** every one can access
- **Simple,** used in most file systems

**3. Capabilities:** store the matrix by rows  
Associated with each user. Indicate which file may be accessed and in what ways.

- Store a list of (object, privilege) pairs with each user. Called capability list.
- **Default:** no one can access.
- **Used** in systems that need to be very secure.
- **Difficult** to share information.

## Input output systems

- **I/O hardware**
  - **I/O device:**
    - \* **Block devices:**  
Store information in fixed size blocks.  
Each block has its own address.
    - \* **Character devices:**  
Deliver or accept a stream of characters.
  - **Device controller:**  
Electronic component of an I/O device, the interface between I/O device and O.S.
    - \* Each controller has a few registers (I/O ports) used for communicating with CPU. Data registers and control/status registers.
    - \* O.S. performs I/O by writing commands into the controller's registers. Then I/O and CPU can work currently. When I/O is done, controller causes an interrupt.

- **I/O software**

- **Five layers**

1. **User processes**

Make I/O call, format I/O, spooling.

2. **Device-independent software**

Naming, protection, blocking, buffering, allocation

3. **Device drivers**

Set up device registers, check status

4. **Interrupt handlers**

Wake up device driver when I/O completed

5. **Hardware**

Perform I/O operation

- **Device driver**

- **Disk arm scheduling algorithms**

- **First-come, first-served (FCFS)**

- \* **Disk arm's motion is based on cylinder request sequence.**

- \* **Example: 40 cylinders, current arm position: 11.**

**Cylinder request sequence:**

**11, 1, 36, 16, 34, 9, 12**

**Motions:**

**10, 35, 20, 18, 25, 3**

**Total seek motions: 111 cylinders.**

- \* **Problem: long average seek time.**

– **Shortest seek first (SSF)**

- \* Always handle the closest request next, minimize seek time

- \* For the above example:

Satisfy sequence: 11, 12, 9, 16, 1, 34, 36

Motions: 1, 3, 7, 15, 33 and 2

Total 61 cylinders.

- \* **Problem:** the arm will tend to stay in the middle of the disk under heavy load.

– Elevator algorithm

- \* Keep moving in the same direction until there are no more outstanding requests in that direction, then switch directions.

- \* For the example:

Satisfy sequence: 11, 12, 16, 34, 36, 9, 1

Motions: 1, 4, 18, 2, 27, 8

Total : 60 cylinders

- \* Good: the total motions fixed for any collection of requests is bounded by twice the number of cylinders.