File System Interface (cont)

- Attributes
  - Name
  - Identifier (internal representation)
  - Type (implicit or explicit)
  - Location (on the device)
  - Size
  - Protection
  - Time, date and user ID
- Operations: create(), open(), read(), write(), seek(), delete(), truncate(), sync()
- Directory structure, flat, hierarchical
- Distributed file systems

File system implementation

- Mounting: Make file system known in the file space
- Partitions: Logical partitioning of disks
- Virtual file system: Hide underlying disk structure
- Allocation: Contigous
  - External fragmentation
  - Defrag programs?
- Linked allocation, Indexed allocation. Unix file system inode interface: direct blocks plus multilevel hierarchy
- Log structured for crash recovery
- Distributed FS

Administrative

- Course project ideas
  - You know how to modify kernel parameters.
  - Ideal project:
    1. First choice: take a significant piece of software code that you use for your research, run it in your target machine (no XP), monitor how it is working (whether it is using as much CPU, disk, memory as you thought you wanted as a programmer), and try to fix it. Show that your fix did not break “everything else” or “why it is okay to break everything else”. Report your experience
    2. Else: Do the same for someone else program.

File system trace papers

- **A trace-driven analysis of the UNIX 4.2 BSD file system** J. K. Ousterhout, H.D. Costa, D. Harrison, J.A. Kunze, M. Kupfer, and J.G.Thompson. In Proc. of the Tenth ACM symposium on 15 Operating System Principles, Dec. 1985
  - Summary: Yingxin, For: Deborah, Against: Kevin
- **Measurements of a distributed file system** Mary G. Baker, John H. Hartman, Michael D. Kupfer, Ken W. Shirriff and John K. Ousterhout, Proceedings of the thirteenth ACM symposium on Operating systems principles, 1991
  - Summary: David, For: Paul, Against: Philip

Trace driven analysis of the UNIX file system

- Rather old, but seminal. Influenced much of file system design for a long time

- Studies like these are extremely important to understand how typical users are using a file system so that you can tune for performance
  - Measuring system performance without perturbing the system is challenging. Drawing the right conclusions (conclusions which are obvious, not obvious, and correct) is also challenging.

- The key is to trace the “typical user population”.
  - Academics do not have access to commercial work loads
  - Chicken and Egg syndrome: Users perform certain tasks because current systems perform poorly.
    - E.g. users may backup their work into a separate file every so often because of poor consistency guarantees.
    - UNIX vi editor saves files by deleting old file, creating a new file with the same name, writing all the data and then closing. If the system crashes after creating and write, before close, data is left in buffers which are lost, leading to a 0 byte file. It happened a lot and so programs create backup files often.

Important conclusions

- Most files are small; whole file transfer and open for short intervals. Most files are short lived. Caching really works.
- UNIX used files as intermediate data transfer mechanisms:
  - E.g. compiler
    - Preprocessor reads .c file -> .i file
    - CC1 reads .i -> .asm file and deletes .i file
    - Assembler reads .asm -> .o file and deletes .asm file
    - Linker reads .o -> executable and deletes .o file
- One solution: Make /tmp an in-memory file system

```
df /tmp
Filesystem           1k-blocks      Used Available Use% Mounted on
swap                   1274544      1776   1272768   1% /tmp
```

Most files are read sequentially

- UNIX provides no support for structured files
- Applications that provide structured access (data bases) use raw file interface and by-pass operating systems
- Solution:
  - Read-ahead to improve performance

Most file accesses are to the same directory

- UNIX has a hierarchical file system structure
- Typical academic users compile, word process from one directory and so all accesses are to a single directory
- File systems such as Coda, AFS have notions of volumes, cells that capture this
- Does this apply to Windows?

Most files are small

- On a departmental machine with 8 MB of main memory, what else do you expect
- Is it true now with our Netscape, xemacs, IE, Power point etc?
- Relatively, it may still be true. On a 60 GB hard disk, 1 MB file may be “small”