30th March 2009  Lecture # 27  Maryam Bashir

Last Week
- Block Devices
  - Disk Characteristics
  - RAID, other block Transformations
- File Systems
  - Semantics
  - Implementation
  - Layout covered
  - Journaling (Today)

Journaling:

For allocating & writing to a block of disk, we have to do following operations:

Update Directory entry

Write Data to block

Update Free space list

The file system on disk may be corrupted due to many reasons, at any point.
So what should be the order of these operations.
If we update free list & then disk file system crashes, then the disk block is lost since we never made entry in directory.
If we write directory entry & file system crashes before free list is updated, then we will have inconsistency since both free list & directory will have entry for that block.

We want to update these as one atomic operation. We can perform these operations as transactions.

If file system crashes before this point, then system will be in consistent state since both directory & free list will not be changed.

We can use database technique of write ahead logging.

We write record of each operation in journal.

T1 □ update dir

T1 □ update free

T1 □ commit
We make sure that each journal is written to disk before update. So if a crash occurs, we can check journal to see which updates took place.

In this way, there is no operation that has not been logged, and completed but if an operation is incomplete the log will have entry for that operation, so system can be recovered to original state or the operation can be completed.

Naive way to do this journal will make it grow out of bounds. At some point where we know some operation has committed, we can remove journal entries for that operation.

**Log Structured File System:**

It has same idea as journaling. The entire file system is a journal.

```
  □  □  □
 □ data □ update □ update
 □ directory □ parent
```
The advantage of this system is that write speed is fast, since write always occur contiguously. The disadvantage is that we need garbage collector for this.

It is also useful for taking back-ups. It is more efficient than doing block-level snapshots.

**Journalized File System:**
1. Log meta-data updates
2. Commit them to disk

**On Reboot:**
Replay any committed but incomplete transactions.

**Log Structured File System:**
Entire File System is a journal.
Security

Two aspects of security:
1. Authentication - who are you
2. Authorization - what are you (allowed to do)

Authorization:

Separate side of security is
- Vulnerability to attacks
- Defense against them

These will be discussed in next lecture

Dimensions of Authorization:

\[ \text{[user]} \xrightarrow{} \text{[objects]} \xrightarrow{} \text{[operations]} \]

Two Approaches of Authorization:

1) Permissions
   \[ \text{objects} \Rightarrow \{ \text{user, op} \} \]
2) Capabilities
   \[ \text{users} \Rightarrow \{ (\text{object, op}) \} \]
Example 1

We have a bunch of documents.

We want different documents to be accessed by different users. The permission model is a more natural fit for this example.

Example 2

System Operations:
- reboot
- mount
- direct disk access

These fall into capability model.

Traditionally, Unix has only used permission model.

Two Ways of Doing Permissions:

1) Owner/Op → Traditional Unix
2) ACLs → Access Control List

Object: has an owner ID, group ID, and permissions

```
ops = {read, write, execute}
```

ops can be read, write, execute...
Matrix for User and File Permissions:

<table>
<thead>
<tr>
<th>Files</th>
<th>U1</th>
<th>U2</th>
<th>U3</th>
<th>U4</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>rw</td>
<td>r--</td>
<td>rw</td>
<td>---</td>
</tr>
<tr>
<td>F2</td>
<td>rw</td>
<td>r--</td>
<td>rw</td>
<td>r--</td>
</tr>
<tr>
<td>F3</td>
<td>r--</td>
<td>rw</td>
<td>r--</td>
<td>r--</td>
</tr>
<tr>
<td>F4</td>
<td>r--</td>
<td>r--</td>
<td>---</td>
<td>r--</td>
</tr>
</tbody>
</table>

Group 1: G1: U1, U4
Group 2: G2: U2, U3
* : anyone

Permissions

File 1: G2: rw--
File 2: G2: rw--
File 3: U2: rw
File 4: G1: r--

allowed (user, op) = owner $\land$ op $\in$ owner.ops
V user $\in$ grp $\land$ op $\in$ group.ops
V op $\in$ world $\land$ op

($\exists$ users $\not\in$ allow/deny)

* deny
$f_1: U_1, x \rightarrow \text{allow}$

$\text{delay:} U_2, y \rightarrow \text{allow}$

$U_3, yu \rightarrow \text{allow}$

* \rightarrow \text{delay}