Lecture #21: File system naming + location – putting it all together

****
Review -- 1 min
****

fs data layout – how to find blocks of a file given its header
- trees, linked lists, etc
- how to get good sequential layout

Transactions –
ACID
Logging (redo, undo, commit, rollback)

****
Outline - 1 min
****

Kernel data structures
Open/close/read/write v. mmap
project overview
Scheduling

****
Preview - 1 min
****

M – midterm
Then, memory systems, protection

****
Lecture - 20 min
****

1. Disk scheduling

Disk can only do 1 request at a time; what order do you choose to do the requests

if 0 or 1 request queued, easy
>1 – try to arrange requests in some order that reduces seek time

1.1 FIFO

QUESTION: how will this work?
Fair among requestors, but order of arrivals may be random → long seeks

1.2 SSTF – shortest seek time first

pick the request that is closest on disk (although called SSTF, today include rotational delay b/c rotation can be as long as seek)

QUESTION: how will this work

good at reducing seeks
can cause starvation

Is it optimal?

1.3 SCAN

SCAN implements elevator algorithm – take the closest request in the direction of travel
No starvation, but retains flavor of SSTF

1.4 CSCAN

2. Kernel data structures for file system

2.1 Read/write interface

Kernel maintains per-process **open file table --**
each entry -- pointer OpenFile object stored in kernel memory

system call (user)	kernel action
open("path")	→ put a pointer to right file in FD table; return index

close(fd) → drop entry from fd table

read(fd, buffer, length) → user refers to open files with index
write(fd, buffer, length) of file descriptor table

What needs to be in OpenFile object to support read/write?
- Inumber (or, if caching, pointer to in-memory FileHeader object)
- per-open-file data (e.g., file position, …)

Why have a separate fd table
- why not just give user pointer to FileHeader object in kernel?
  - how does kernel know when it can free object?
  - convenience: per-open-file data (file position, …)
- why not just use path for all operations (e.g., read(path, offset, …))
  - efficiency – string operations, protection checks

2.2 Caching

Read and write end up calling disk block read/disk block writes

We've stated several times that we need good caching for file systems to work well. How does this work?

Simple answer: block cache

Replace all uses of
```
ReadDisk(blockNum, buffer)
```
With
```
ReadDiskCache(blockNum, buffer){
    ptr = cache.get(blockNum); // just a hash table
    if(ptr){
        copy BLKSIZE bytes from ptr to buffer
    }
    else{
        newBuf = malloc(BLKSIZE);
        ReadDisk(blockNum, newBuf);
        cache.insert(blockNum, newBuf);
        copy(blockNum, buffer, BLKSIZE);
```



```
    }
}
```

Advantage: simple – write all FS code as if always reading from disk and insert the cache at the lowest level

Issues: replacement policy --> in a few weeks when we talk about memory systems

Disadvantage: copy overhead – each read copies block into a new buffer

For in-kernel use, we could return a pointer to cached version
- More complex: need to deal with reference counting, etc., but we could make it work…

What about avoiding copies to user space?

2.3 Mmap interface

void *mmap(int fd, size length, …)

map the specified open file into a region of my virtual memory, and return a pointer to that region

How might we implement this?
How would we update your page table?
How do I read a file?
How do I write a file?
What happens if a page is evicted from the cache?
What happens if a page is brought back into the cache?

****
Admin - 3 min
****

Project out. Start early.

****
Lecture - 23 min
****

3. project 2-3 –

It looks like a lot
It will force you to **understand** file systems
Once you do, it is simple if you **follow a modular design**

3.1 atomic disk

```
xid = adisk->beginTransaction()
error = adisk->writeSector(xid, secNum, data)
error = adisk->writeSector(xid, secNum2, data2)
error = adisk->readSector(xid, secNum, data)
error = adisk->commitTransaction(xid);
```

What should writeSector do?
What should readSector do?
What should commitTransaction do?

Concurrency:
Option 1: (recommended) only allow one active transaction at a time. (What should beginTransaction() do? What should endTransaction() do?)
Option 2: allow multiple outstanding transactions. (What should beginTransaction() do? What would endTransaction() do? What would readSector() do?)

3.2 Tree

Each tree will correspond to a file.

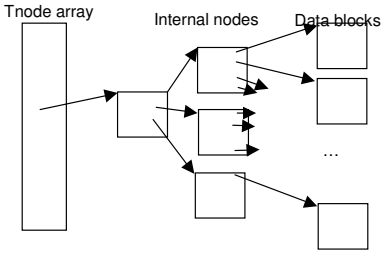
Possible milestones
- design data structures; pseudo-code methods
- format disk; test
- create/delete tree; test
- read/write small trees; test

- read/write large trees; test

```
Xid = ptree->beginTrans();
tnum  = ptree->createTree(xid);
Error = ptree->writeData(xid, tnum, 0, data);
Error = ptree->writeData(xid, tnum, 1, data1);
Error = ptree->readData(xid, tnum, 0, data2);
Error = ptree->commitTrans(xid);
```

Slightly different index structure than any we showed in class – similar to multi-level index, but grow tree dynamically

What is on adisk? tnode array, data array.



Keep track of invariants on nodes in tree and it will stay simple.
Try to write code before you've *understood* invariants, and it will be very complex.

pseudo-code for readData (no error checking, …)

```
Ptree::readData(xid, tnum, blockId, buffer){
        tHdr = getTreeHeader(xid, tnum);
        sector = tHdr->getBlkAddr(xid, blockId);
        adisk->readSector(xid, sector, buffer);
        }
}
```

What does getTreeHeader do?

What does TreeHeader::getBlkAddr(xid, blockId) do?
How would you create/delete a file? (How find a free thdr?)
How would you format the disk?

3.3 flat file system – read and write using inumbers and offsets

not much to add above tree storage…

3.4 directory-based file system

– add names, directories, and open file table
What is pseudo-code for read?

```
read(fd, start, length, buffer){
        xid = flatFS->beginTransaction();
        of = fdTable.getOpenFile(fd);
        inum = of.getINum();
        flatFS->iread(xid, inum, start, length, buffer);
        flatFS->endTransaction(xid);
}
```

What is pseudo-code for open? (Key idea: recursion to find inum given path name).

3.5 pitfalls

test as you go; get part 1 completely working and bug free before starting part 2; etc.

casting – array of bytes (from disk) into objects (in memory)
Suggest – constructor that takes a sector number
e.g., InternalNode::InternalNode(Adisk disk, TransId xid, int sector);

partner issues – see handout on suggestions for how to manage partnership

****
Summary - 1 min
****