CSE 451: Operating Systems

Section 8

Linux buffer cache; design principles

Outline

- Thread preemption (project 2b)
- RPC
- Linux buffer cache
- Networking design principles

Mutex: no preemption

```
mutex_lock(lock)
  if (lock->held == 0)
    lock->held = 1;
  else
    enqueue(
      lock->waitq,
      current);
    schedule();
```

```
mutex_unlock(lock)
  if (!is_empty(
    lock->waitq))
    next = dequeue(
      lock->waitq);
    enqueue(readyq,
      next);
  else
    lock->held = 0;
```


Mutex: with preemption (wrong)

```
mutex_lock(lock)
  while(test_and_set
    (lock->held))
    {}
```

```
mutex_unlock(lock)
  atomic_clear(
    lock->held);
```

Mutex: with preemption (wrong)

```
mutex_lock(lock)
  while(test_and_set
    (lock->held))
    schedule();
```

```
mutex_unlock(lock)
  atomic_clear(
    lock->held);
```


Mutex: with preemption (wrong)

```
mutex_lock(lock)
  while(test_and_set
    (lock->held))
    enqueue(
      lock->waitq,
      current);
    schedule();
```

```
mutex_unlock(lock)
  if (!is_empty(
    lock->waitq))
    splx(HIGH);
    next = dequeue(
      lock->waitq);
    enqueue(readyq,
      next);
    splx(LOW);
  atomic_clear(
    lock->held);
```


Mutex: with preemption (wrong)

```
mutex_lock(lock)
  while(test_and_set
    (lock->held))
    enqueue(
      lock->waitq,
      current);
    schedule();
```

What if prempted here?

```
mutex_unlock(lock)
  if (!is_empty(
    lock->waitq))
    splx(HIGH);
    next = dequeue(
      lock->waitq);
    enqueue(readyq,
      next);
    splx(LOW);
  atomic_clear(
    lock->held);
```


Mutex: with preemption (right)

```
mutex_lock(lock)
  while(test_and_set(
    lock->codelock))
    {}
  if (lock->held == 0)
    lock->held = 1;
    atomic_clear(
      lock->codelock)
  else
    enqueue(
      lock->waitq,
      current);
    atomic_clear(
      lock->codelock)
    schedule();
```

```
mutex_unlock(lock)
  while(test_and_set(
    lock->codelock))
    {}
  if (!is_empty(
    lock->waitq))
    next = dequeue(
      lock->waitq);
    splx(HIGH);
    enqueue(readyq,
      next);
    splx(LOW);
  else
    lock->held = 0;
  atomic_clear(
    lock->codelock)
```

RPC

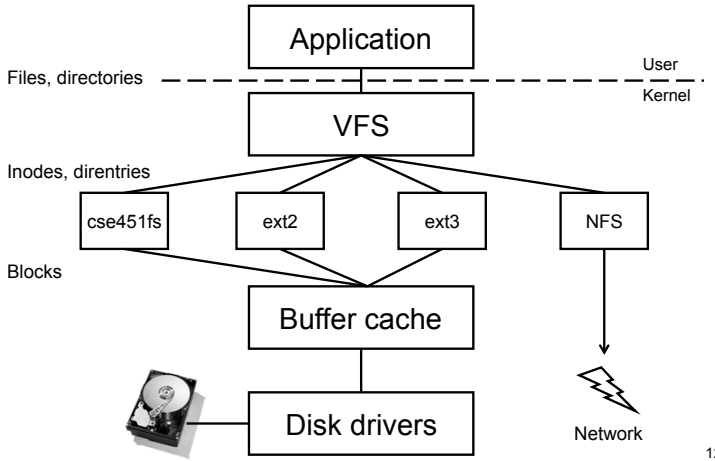
RPC

- Remote procedure call: causes a procedure to execute in some other *address space*
  - Usually an address space on some other machine
- Interface description language (IDL) defines the interface that the server makes available to the client

RPC on Android

- Android uses RPC for communication between applications and system components
  - All on the same device!
- Uses all of the standard RPC components:
  - IDL file
  - Auto-generated stubs for client, server
  - Marshalling and unmarshalling of arguments…

Linux file system layers



Linux buffer cache

- Buffer cache: just an area of memory
  - `cat /proc/meminfo`
- Caches disk blocks and buffers writes
  - File `read()` checks for block already in buffer cache
    - If not, brings block from disk into memory
  - File `write()` is performed in memory first
    - Data later written back to disk (when? By who?)

Linux buffer cache

- Block is represented by a `buffer_head`
  - Actual data is in `buffer_head->b_data`
  - Cache manipulates `buffer_head` and its `b_data` separately

Buffer cache interface

- `include/linux/buffer_head.h`
- Read a block: FS uses `sb_bread()`:
  - Find the corresponding `buffer_head`
    - Create it if it doesn't exist
  - Make sure `buffer_head->b_data` is in memory (read from disk if necessary)

Buffer cache interface

- Write a block: `mark_buffer_dirty()`, then `brelse()`
  - Mark buffer as changed and release to kernel
  - Kernel writes block back to disk at a convenient time
    - *bdflush* / *pdflush* threads
    - `sync` command

cse451fs functions

```
cse451_bread(struct buffer_head **pbh,
  struct inode *inode, int block, int create)
```

- Gets `buffer_head` for given disk block, ensuring its `b_data` is in memory and ready to use
- Calls `cse451_getblk()`

```
cse451_getblk(struct buffer_head **pbh,
  struct inode *inode, int block, int create)
```

- Gets `buffer_head` for given disk block, creating it if it doesn't exist
- Doesn't necessarily bring `b_data` into cache

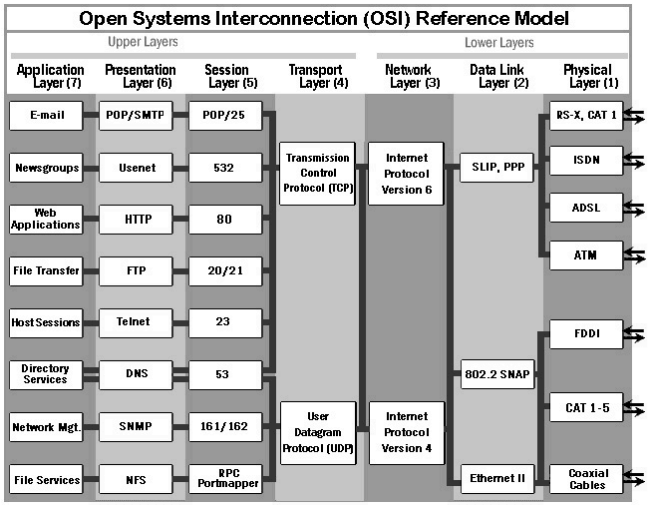
* Both of these functions increment the block's refcount, so must be paired with a `brelse()`

Project 3

* Questions?

Networking design principles

* A few key principles:
  - Layering
  - End-to-end principle
  - Separation of mechanism and policy
* All of these apply to operating systems (and elsewhere!) as well



Layering

- Internet designers didn't get it all right the first time
- Design for choice
  - Rigid designs will be broken

End-to-end principle

- Danger of putting functionality at lower layers: upper layers won't need it, but will pay cost anyway
  - Example: reliability checksums
- E2E principle says to move functionality towards upper layers (closer to application)
- Other ways of phrasing it:
  - Smart endpoints, dumb network
  - Application knows best what it needs

Mechanism vs. policy

- Principle: design system so that mechanisms are separate from policy
  - Implement mechanisms that enable wide range of policies
  - Caveat: usability ("burden of choice")