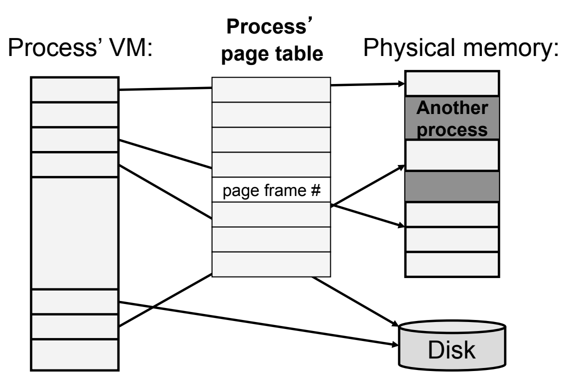
CSE 451: Operating Systems

Section 9:
Storage; networks

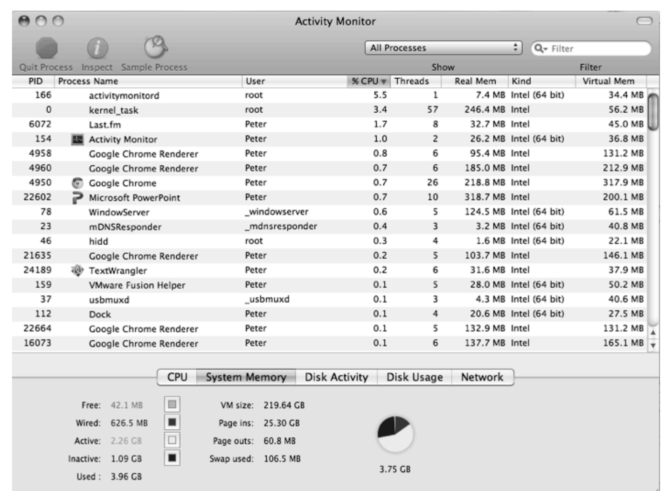
Outline

- ∗ Virtual memory
- ∗ Secondary storage
- ∗ File systems
- ∗ RPC
- ∗ Networks

Virtual memory



(slide from Chernyak Fall 2009)



Page replacement algorithms

- Belady's algorithm
  - Replace the page that's going to be needed farthest in the future
- FIFO (First In/First Out)
  - Replace the oldest page with the one being paged in
  - Not very good in practice, suffers from Belady's Anomaly
- Second-chance (modified FIFO)
  - FIFO, but skip referenced pages
  - VAX/VMS used this
- Random
  - Better than FIFO!
- NFU (Not Frequently Used)
  - Replace the page used the least number of times
- LRU (Least Recently Used)
  - Replace the least-recently used page
  - Works well but expensive to implement
- LRU Clock (Modified LRU)
  - Replace the least recently used page, with a hard limit on the max time since used

Example of Belady's anomaly

Sequence of page requests:	3	2	1	0	3	2	4	3	2	1	0	4
3 physical page frames:	3	3	3	0	0	0	4	4	4	4	4	4
		2	2	2	3	3	3	3	3	1	1	1
			1	1	1	2	2	2	2	2	0	0

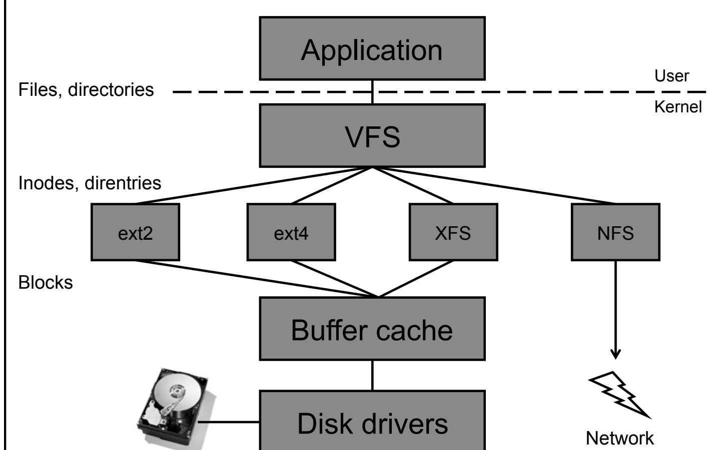
Page faults (in red): 9

Example of Belady's anomaly

Sequence of page requests:	3	2	1	0	3	2	4	3	2	1	0	4
4 physical page frames:	3	3	3	3	3	3	4	4	4	4	0	0
		2	2	2	2	2	2	3	3	3	3	4
			1	1	1	1	1	1	2	2	2	2
				0	0	0	0	0	0	1	1	1

Page faults (in red): 10

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?)
    - Kernel writes block back to disk at a convenient time (*flush* threads), or synchronously if user requests it

Is flash the answer to all of our storage problems?

- Do solid state flash drives obviate the need for the buffer cache?
- NAND flash technology faces scaling challenges that may be insurmountable
  - As density / capacity increases, all other important characteristics are degraded: latency, write endurance, energy efficiency
    http://www.theregister.co.uk/2012/02/21/nand_bleak_future/
    http://cseweb.ucsd.edu/users/swanson/papers/FAST2012BleakFlash.pdf

Project 3: tools

- **hexdump**
- **dumpe2fs**
- **valgrind**
- `mkFilesysFile.sh`

Project 3: tips

- Use the ext2fs.h and ext2_types.h header files
  - Most structs are already defined for you
- Don't use absolute values in your code
  - Use constants from the header files
  - Look up values in the superblock, then calculate other values that you need

Project 3: tips

* `fileIOExample.c`
  * Notice that it checks the return value of every system/library call – you must do the same!
* Don't forget to set timestamps of recovered files
* Follow the turnin instructions
  * Don't change filenames, etc.
  * Disable debugging printfs before submission

Bit operations

* Remember how these operators work in C:
  * `/ % & | << >>`
* Given an inode number, how do we find the right byte in the inode bitmap?
  * Hint: use `/`
* Given a byte in the bitmap, how do we check if the inode's bit is set?
  * Hint: use `%`, `<<`, `&`

Project 3: testing

* How will you test your undelete program?
  * Ideas:
    * Delete small / large files
    * Use small / large file systems
    * `mkFilesysFile.sh`: the **mkfs.ext2** command inside takes many options; your undelete program should still work if basic options are changed!

Project 3

* Questions?

RPC

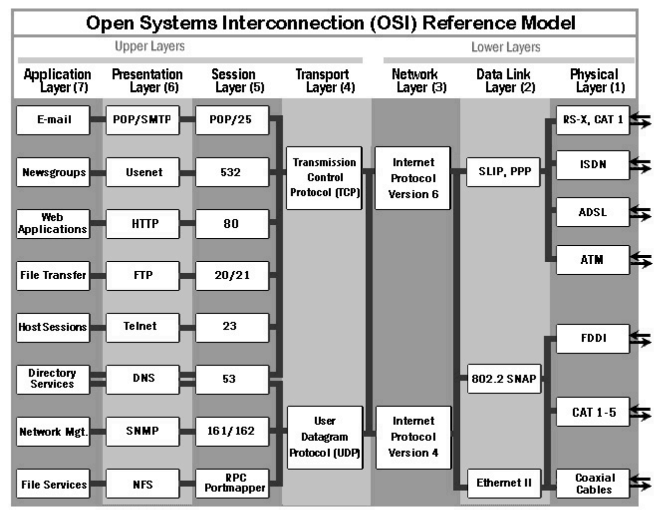
* Remote procedure call: causes a procedure to execute in some other *address space*
  * Usually an address space on some other machine
* Interface description language (IDL) defines the interface that the server makes available to the client

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…

Networking design principles

* A few key principles:
  * Layering
  * Encapsulation
  * End-to-end principle
* 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