Lecture #1, Monday April 1, 2019

Bring Notecards and White board markers

1. Introductions
    1. Who I am
    2. TA's, William Ceriale, Logan Weber, Dilraj Devgun, Vinny Palaniappan
    3. My story
        i. UW student 75-84, CSE 451 when it was first offered
        ii. DECwest (5 years)
        iii. Microsoft (12 years), Windows NT, and File systems
2. Class details and organization
    1. Class Prerequisites: CSE332: Data Structures and Parallelism, CSE333: Systems Programming, CSE351: The Hardware/Software Interface
    2. Lectures, textbook, exams, and projects (work in teams)
        i. Lecture format, no powerpoint, interactive, joint discovery/discussion, open to new ideas, 2 truths and a lie, plan to set aside a few hack weeks w/o lectures.
        ii. Project will be online soon with due dates.
    3. Grading projects 70%, exam 30% (plus possible written assignments)
    4. See webpage for Calendar and links to canvas and gitlab
3. Notecard exercise, No name, Mini-tweet size
    1. What is and OS
    2. One thing that you hope to learn from this class
4. Course Objectives and strategy
    1. (One thing that I hope you learn from this class) Now an OS is designed and built. To better know how to use the OS
    2. Debugging large programs, adding new features to an existing (incomplete codebase)
    3. Quote in the Cutler lab "Bugs: if you don't put them in, you don't have to take them out."
    4. Drinking from the firehose for the first few weeks
    5. Textbook: academic head knowledge
    6. Lectures: enhance and supplement the textbook and do some deep dives into some specific implementation issues.
    7. Projects: get hand dirty, learn by doing

5. Course Roadmap
    1. Chapter 2 Kernel Abstraction
    2. Chapter 4 Concurrency
    3. Chapter 5 and 6 Synchronization
    4. Chapter 7 Scheduling
    5. Chapter 8 and 9 Memory Management
    6. Not in textbook Basic I/O organization
    7. Chapter 11, 12, 13, 14 Storage
    8. Wrap up loose ends
6. Mission statement of an OS
    1. Class exercise, together write an OS mission statement (e.g., mission statement for the UW CSE department might be to prepare students in the art of computer usage for the coming century.)
    2. Provide a means for programs to effortlessly utilize the capability of the computer system
7. OS Roles
    1. Referee (resource allocation and isolation)
    2. Illusionist (one system, many applications/users)
    3. Glue (intentional interaction between application and services)
8. OS Challenges
    1. Usual list of providing Reliability, Availability, Security, Privacy, Portability, Performance
    2. Challenge that is not really talked about is forward compatibility with evolving HW and System requirements, e.g., word size 8 – 16 – 32 – 64. Memory size, disk size
9. OS History
    1. Batch to timeshare
    2. Single User to multiuser to single user (PC) to multiuser (servers, etc.)
    3. Cost of computer time compared to people time
    4. Single processor to multiple processors to distributed systems

Lecture #2, Monday April 3, 2019

10. (Chapter 2) Kernel Abstraction
    1. Roadmap for next two days
        i. Hardware modes
        ii. Interrupts, exceptions, and syscalls (traps)
        iii. Memory layout
        iv. Processes
        v. Booting the OS
    2. Dual modes (2.2) of operation (controlled via EFLAGS on x86 or status register on MIPS).
    3. Kernel mode
        i. Full access and privilege to access all the HW
            1. e.g., read and write any memory location or I/O device).
            2. Can also halt the processor.
    4. User Mode
        i. Limited privileges.
        ii. What instructions would be off-limits in user mode?
    5. Are two modes the right number?
    6. Is one mode the right number?
    7. Quick look at memory protection to explain user mode limitations
    8. Switching modes (2.3, 2.4)
        i. From user to kernel
            1. Exceptions
            2. System calls (traps)
            3. Interrupts
        ii. From kernel to user
            1. Start a new process/thread
            2. Return from Interrupt, exception or system call
            3. Process/thread context switch
            4. User level upcall (Unix signal)
        iii. Communication from Physical devices to Kernel
            1. Interrupts (as opposed to polling, DMA makes things run faster)

               Types/reasons for interrupts, I/O completion, timer,

9. Dissect the intricacies of an interrupt, an exception, and a system call
    i. Start of an interrupt and return from an interrupt
    ii. Interrupt vector (slides)
    iii. Interrupt stack
    iv. Interrupt masking, and interrupt levels
    v. Interrupt handlers
    vi. Polling as an alternative to interrupts

11. Quick recap
    1. Kernel (core) and User Mode
        i. Syscalls/traps
        ii. A means for user mode code to request a system service (needs parameter checking, etc.)
        iii. Like a procedure call with the return to the next instruction
        iv. Instead of a set address like a function call it uses an index into a system services table (i.e., interrupt vector)
        v. Cost of a system call
    2. Exceptions
        i. Unexpected/unanticipated result of a user action/instruction e.g., divide by zero, or illegal memory address
        ii. What to do with the exception and where to restart the user if at all
    3. Interrupts
        i. External event that needs attention of the CPU/processor
        ii. Happens between instructions so that we can restart the interrupted instruction. Transparent to the one being interrupted
        iii. Interrupt level, masking, turning off interrupts, nested interrupts
        iv. Question/quagmire: interrupting the interrupter
        v. Polling to also handle devices and DMA
        vi. Upcalls for user mode interrupts

12. Virtual Memory Layout
    1. Simple base limit registers to understand the problem (slides)
    2. Move to virtual memory (a flat address model) with named segments

3. Address space map i.e., kernel and user space

Lecture #4, Monday April 8, 2019

13. Deep dive: booting the OS
    1. Physical memory and what is on disk
    2. Sector 0 (bootasm.s, 16 bit real mode, boot block)
    3. Boot loader (bootmain.s, 32 bit, ntldr)
    4. OS kernel (main.c ntoskrnl.exe)
14. How OS loads a program and resolves library references
    1. Monolithic system
    2. DLL function
15. Process Abstraction (2.1)
    1. What is a process, what does it contain, and how is it used
    2. Unit of Ownership in the OS,
        i. privileges, capabilities
    3. Resources
        i. Memory
        ii. Handles to system resources e.g., files and events
    4. Traditionally a single execution state but we'll jump straight to multiple threads of execution per process

Lecture #5, Wednesday April 10, 2019

Roadmap

- Process management
- Nitty Gritty: PCB, TCB/TLS
- Kernel threads and user threads
- Communication between threads and between processes
- Maybe pipe example before going into synchronization

16. Process management
    1. Distinction between multi-processing and multi-processor
    2. Creation, deletion
        i. Fork
        ii. Exec
        iii. Wait or join
        iv. Exit
        v. Signal
    3. Inheritance
    4. Draw diagram of process as boxes with threads and stacks on top of a CPU
    5. Work through process creation in detail, starting with the shell
    6. Scheduling
        i. Context switch (yield model [voluntary or cooperative] versus time slicing or multiplexing [involuntary])
        ii. Policy versus mechanism
        iii. Lifecycle, init – ready – running – waiting – exit
        iv. Idle thread

Lecture #6, Friday April 12, 2019

Think of a final exam question.

17. Step back for a look at the big picture of the class covers
    1. Process care and feeding (with occasional segues into HW support)
    2. Process isolation but also communication
    3. Process accessing system resources (CPU i.e., scheduling)
    4. Process using memory
    5. Process doing file I/O
18. How to communicate between processes
    1. Signals
    2. Files
    3. Pipes
        i. One way – consumer producer
        ii. Two way – client server
19. Deep dive into pipes
    1. Deep dive: Inter-Process Communication (Producer-Consumer Communication)
    2. From a user command line viewpoint
        i. > echo “hello world” | wc
        ii. > echo “hello world” > /tmp/xyz; wc < /tmp/xyz
    3. Implementation
        i. Two file handles reader (consumer), writer (producer)
        ii. Buffer in kernel space, some options
            1. Fixed size array, but how big
            2. Dynamic allocation, but how much, introduce the notion of quotas
        iii. What about synchronization access to the buffer
        iv. Byte pipes or message pipes
        v. Named pipes versus unnamed pipes
        vi. Mailbox in windows for networked pipes

Next up Chapter 5 and 6 synchronization

XK uses spinlocks and sleeplocks and channels not wait queues

20. (6) (Chapter 5) Synchronization
    1. Keep using Pipe construction as a motivation for synchronization
    2. Challenges
    3. Shared Objects
    4. Some definitions
        i. Race Condition
        ii. Mutual Exclusion (critical section)
        iii. Lock – guarantee mutual exclusion
            1. Acquire and release

5. Grab bag of synchronization primitives
    i. REMEMBER the operations we have ACQUIRE/LOCK and RELEASE/UNLOCK
    ii. Keep list of what is available in user or kernel mode, advantages and disadvantages or each.
    iii. We are doing this just to get pipes to work correctly. Build a pipe picture with shared data (buffer, tail, head) and a lock.
    iv. Disable interrupts on uniprocessor systems (only in kernel)
        1. List advantages and disadvantage of disabling interrupts
    v. Spinlocks
        1. Busy signal on phone calls as an analogy
        2. Needs HW support xchg and lock prefix on multiprocessor x86 systems
            a. Finish xchg in spinlock example, emphasize that xchg is hardware atomic. Lock Prefix for MP systems
        3. XK has spinlocks
        4. List advantages and disadvantage of a spinlock
    vi. BOTH interrupt finagling and true spinning are usually hidden in the SPINLOCK call, later when we talk about scheduling, memory management we'll see implications of this.
    vii. Sleeplock in XK has sleep/wakeup for blocking/yielding synchronization
        1. Being put on hold as an analogy

Lecture #8, April 17, 2019

Finish up with Mesa and Hoare semantics

Talk about monitors, good and bad points.

viii. Other locks that block
    1. Events
    2. Semaphore (binary semaphore, mutex)

6. Deep dive: blocking bounded Queue (Pipes)
    i. Figure 5.8
    ii. Windows does not use a bounded buffer, but how to avoid using up all of memory
7. Finish pipes to the bitter end
    i. One way – consumer-producer
    ii. Two way – client server
    iii. Buffer in kernel space, some options
        1. Fixed size array, but how big
        2. Dynamic allocation, but how much, introduce the notion of quotas
    iv. Byte pipes or message pipes
    v. Named pipes versus unnamed pipes (fifo in linux)
    vi. Mailbox in windows for networked pipes

Lecture #9, April 19, 2019

Locks upon Locks: Mention how higher order locks are often implemented using spinlocks

8. Deep dive: Reader/Writer Lock (EResource)

Motivate the need for both shared (reader) and exclusive (writer) access.

Work as a class on a high level design

Implement InitializeReadWriterLock(Lock);

AcquireShared(Lock);

AcquireExclusive(Lock);

Release(Lock);

Things to consider

- Starve exclusive (yes or no?)
- Release for someone else
- Recursive acquisition

Lecture #10, April 22, 2019

Almost finished with locks

- Mentioned that more complex locks still require simple locks to synchronize internally
- Reader/Writer locks – more changes to come with deadlocks, scheduling (priority inversion)
- Hacks added to locks in aid in system development
    - Bug check if returning while holding a lock
    - Storing owning PID, and return address
- Locks are used to guarantee the correctness of the code
- There is a tension when using locks between making the code correct and making it efficient
- Efficiency involves parallelism,
    - Lock granularity
    - How long a lock is held

9. Uniprocesser versus multiprocessor

i. Optimizations around the way multiprocessors work with memory especially

10. MCS Locks (Chapter 6.3.1)

Fixes memory contention issue with spinlocks in multiprocessor systems.

11. RCU locks (Chapter 6.3.2)

Lecture #11, April 24, 2019

Next three lecture topics

Wednesday – Deadlocks (Section 6.5, pp. 285-305)

Friday – Threads (Back to Chapter 4)

Monday – Scheduling (Chapter 7)

21. Deadlocks

1. Four conditions for a deadlock
    i. Bounded resources
    ii. No preemption
    iii. Wait while holding
    iv. Circular Wait
2. Deadlock is not starvation per se but the symptoms might look similar
3. Quick solutions (that aren't very practical)
    i. Unlimited resources
    ii. Allow preemption
    iii. Release before acquiring another lock
    iv. Acquire all locks at once, early on

4. Academic but otherwise uninteresting topics

    i. Deadlock prevention (Banker's Algorithm)
    ii. Detection strategies (Graph traversal and aborting a process)
    iii. Dining Philosophers and directed acyclic graphs

5. Methods used in system development (not always successfully)

    i. Lock ordering
    ii. A lot of careful analysis
    iii. Windows is a deadlock minefield
    iv. The nice thing about debugging deadlocks is that they are someone easy to chase down on a frozen system.
        1. This is where storing lock ownership and return address help

6. Lock Contention

i. Use nonblocking techniques when possible, lock free data structures (e.g., compare exchange to handle a linked list)
ii. Mainly used for hotly contented locks
iii. Rule of thumb: A tension exists. Locks that don't have much contention don't impact performance much. Locks with a lot of contention adversely impact performance. Where you really need the lock, it hurts, but where it isn't as necessary it doesn't cost much.

Lecture #12, April 26, 2019

22. Back to processes and introduce threads

1. PCB
    i. Memory and open handles
    ii. Privileges/
    iii. capabilities, quotas, and affinity, priority
    iv. A lot more accounting information, e.g., kernel and user time
2. TCB & TLS
    i. Introduce multi-threading
    ii. 2 stacks per thread – user/kernel – how big, guard pages, where to put them in a multithreaded process
    iii. Contains
        1. Stack information, context switch information, priority
        2. State, ready, running, waiting, and Wait list
        3. A lot more bookkeeping details, time, affinity,
    iv. Work through thread creation

23. Kernel threads versus User threads

1. Visible to the Kernel meaning
    i. Schedulable
    ii. Can allow others threads in a process to run if one is waiting for I/O
2. Kernel threads are 1:1 mapping between user and kernel
3. Traditional User threads are N:1 mapping between user and kernel
4. User threads have a lot less overhead to do a context switch (no system call)
5. However user mode threads have traditionally not be able to take advantage of the user to do Async I/O. That is been changing recently with Scheduler Activation
6. Scheduler Activation uses N:M mapping between user and kernel

Lecture #13, April 29, 2019

Give last motivating example of Scheduler Activation with a stuck read call

Talk about kernel only threads, and the idle thread (idle process)

Why context switch (What is wrong with good old batch processing)

The Cost of context switching (wasted CPU cycles and invalidated caches)

When to schedule a context switch

24. (3) (Chapter 7) Scheduling
    1. Uniprocessor versus multiprocessor
    2. Performance metrics
        i. Response time
        ii. Predictability
        iii. Throughput
        iv. Scheduling overhead
        v. Fairness
        vi. Starvation
    3. Grab bag of scheduling algorithms
        i. FIFO
        ii. Shortest job first
        iii. Round Robin

Lecture #14, May 1, 2019

Problems with simple Priority Queue (starvation, how to decide priorities)

4. Deep dive: Multi-Level Feedback
    i. Design a model of MLFQ
    ii. Priority inversion
    iii. Idle thread
    iv. Threads always being created and destroyed
    v. Critique of MLFQ (Big hammer for a mostly empty queue?)
    vi. Grade MLFQ based on performance metrics
        1. Response time
        2. Predictability
        3. Throughput
        4. Scheduling overhead
        5. Fairness
        6. Starvation
5. Deep dive:
    i. Process, thread, or user prioritized scheduling
    ii. I/O vs CPU intensive threads
    iii. Priority inversion
6. Linux scheduling: Completely fair scheduling, Min-max fairness
7. How to charge CPU time for interrupt handling?

Lecture #15, May 3, 2019

From Feedback:

- The pattern that I am trying to explain, as best I can, as someone pointed out, is to walk through my own slow thought process to develop these concepts, with the dead ends and gotchas. My goal is to let everyone develop it along with me.
- The final exam question box is an experiment. My thoughts are (1) it would be a useful exercise for you to think about the material, what is important and how you might phrase a question around it (2) we'll use only the appropriate questions with redaction for clarity and brevity and ease of grading and (3) our new thoughts are to use the questions you submit to help with review of the material in class or sections.
- Delaying the discussion on Multiprocessor versus uniprocessor scheduling issues until after MM because MM issues play a large role in this area.

But to finish with the penultimate word on scheduling

8. Thrashing – too many tasks causing too many context switches.
    i. Drive the curve
9. Recap: FIFO, SJF and Shortest Remaining Job, Round Robin, Priority Queues, MLFQ

25. (6) (Chapter 8) Memory Management
    1. Segments (base & limit values) vs. paging
        i. Introduce Internal versus external fragmentation
    2. Address Translation
        i. Physical versus Virtual Memory

Lecture #16, May 6, 2019

How to expand the size of a running program in the segmentation model.

For now all HW design from a bit a pretty high level

Finish up with simple paging example, explain how it solves external fragmentation problem

Develop 32 bit memory model

ii. Review Page table (single and multi-level) , PTE with 4K pages
    1. Size of Page Table in pages 4KB + 1024*4KB = 4,198,400 for a full memory address which is a 0.1% tax
    2. Contents of a PTE: page frame #, Bits – Present, Writable, User, Accessed, Dirty, write-through, cache disable, 3 SW available bits
    3. Windows 10 system with approx. 70 processes -> 280 MB if all full
    4. How big is a page table for a sparse program

Lecture #17, May 8, 2019

Some observations

1. Translation cost
2. A benefit with paging is that all of the program need not be in memory all at once
    a. How much to have in memory to run a program
    b. Paging and the working set model
5. Able to expand program, is this possible (cheap) when using segments

Optimizations

iii. TLB optimization typical size is from 32 to 64 entries, and maybe 2x or 4x set associative
iv. Locality of reference – Spatial and temporal
v. L1, L2, and L3 level caches (on my desktop i7 system, 8GB RAM, 4 cores, 8 logical processors, L1 256KB, L2 1.0 MB, L3 8.0 MB (virtual versus physical cache)
vi. I-cache and D-cache
vii. Inverted page tables
viii. Grab bag of layout strategies

Now how SW is setup up to use the Hardware

Lecture #18, May 10, 2019

Working set is a concept that guides how we approach page management and replacement.

Today's topic Page Replacement.

What page to bring in is somewhat obvious, which page to throw out is less obvious.

3. (Chapter 9) Caching and Virtual Memory
    i. Replacement strategies – which page to choose to remove.
        1. MIN Furthest into the future
        2. FIFO (Belady's Anomaly)
        3. LRU
        4. LRU-Clock (possible to use PTE SW bit to count accesses
        5. Random

Lecture #19, May 13, 2019

Mention use do dirty bit in PTE

ii. Deep dive: Shared Memory and COW's

iii. Inverted page tables

i. Deep dive: Paging in Windows, the state of pages
ii. JZ problem, ask students to figure out what was causing the problem and then how to fix it.
iii. Deep dive: paging kernel memory, paged vs. nonpaged pool

i. Deep dive: Memory performance test part 1 zeroed pages

Lecture #22, May 20, 2019

Bring in slide rule

   ii. Deep dive: Memory performance test part 1, TLB performance hump
   iii. page coloring
   iv. Deep dive: Memory performance test part 2, L2 cache
4. Paged kernel. Paged and nonpaged pool, and all sort of gotchas
   i. ExAllocatePool, kmalloc

Lecture #23, May 22, 2019

Final exam style questions. Mostly multiple choice style. For example, I might remind you of the test program we just finished talking about. And ask which best describes the timing characteristics.

a. A good answer
b. The good answer but minus an important feature
c. The good answer plus too much fluff
d. A plausible answer but not as good as the good answer
e. Just a lot of mumble jumble
f. All of the above, or none of the above

Roadmap to the end of the quarter (read chapters 11, 12, 13, 14)

1. I/O system from
    a. User level, to
    b. OS/FS software level, to
    c. on-disk structure layout, to
    d. storage media. And
    e. then back out again.
    f. Emphasis will be on Windows internals, because that I know that really well, Linux/Unix is similar and you read about those in the books.
2. Memory mapped I/O
3. Multiprocessor scheduling
4. Spector/meltdown
5. And maybe some loose ends.

Naming in the OS

1. Everything (files, directories, users, events, devices, locks, Memory sections, etc.) is given a unique name in the system
2. Start with usual directories and files
3. Add other kernel objects (devices and put them in the name space)

26. (2) (Not in Textbook) Basic I/O logic

I/O a few aspects to discuss

1. Abstraction for how I/O is presented/modelled to the user and implemented in the OS
2. grab bag of HW devices. Persistent storage, network, local devices such as the keyboard, mouse, video, sound, etc. The goal is to have some consistent model in the OS to handle all these devices. In Windows this is the I/O manager, with things people might plug-and-play, etc. We have specialized HW devices, USB, serial, parallel, etc. for device types to consider.
3. the punch line is that it is not a pretty picture. It is more hacks upon hacks. When we deep dive into the Windows I/O system we'll see this from a very high level.
4. We'll mainly concentrate on persistent storage (hard drives, etc.)

27. (4) (Chapters 11, 12, 13, 14) Storage
    1. User App Level
        i. Files and directories are the two main entities used for persistent storage
        ii. The most common model for a user is to view/treat a file as simply a linear bag of bits of arbitrary sizes
        iii. Naming is important. This includes directory and other location information
        iv. The usual suspect of operations - Create, link, unlink, createdir, rmdir – for manipulating files and directories.
        v. Operations for files themselves - Open, close, read, write, seek
        vi. Operations for directories – search, enumerate
        vii. Synchronous and asynchronous operations. For example, notify me when files in a directory change
        viii. Fsync
    2. OS/FS Software Level
        i. Walk through open, read/write, close, and what is stored internally
            1. Path from Create File call to the I/O, FS, Device driver
            2. Handles and handle table, and Objects in the OS
            3. FCB and CCB
            4. Current working directory, relative opens
            5. Current file offset
            6. Shared opens
            7. Delayed closes
            8. How to handle directories and change notifications

ii. This is where the File System software lives and is integrated into the OS via what called the I/O system (MM and the Cache manager are also here)
iii. Quite often these user level app calls become multiple calls into the OS. Or are overloaded into a single OS call with a ton of options (this is the Windows Model, e.g., the NtOpenFileCall has flags for create if, open if, file, directory, etc)
iv. Identify which opened file/directory to work on is specified using handles which are simply indices into a processes handle table. The handle table then points off to the type of device it denotes, which can be an I/O device or kernel device, like a section or event

Lecture #24, May 24, 2019

A. Give another example why seek may have funny semantics. If two people are manipulating the file. One seek and the other truncates. Now the seek location is out in space.

B. Finish up with how Deleting file are done

C. Hard versus symbolic links

D. Tunneling

3. Quick introduction to hard drives, just enough so that we can talk about file system design
    i. Cylinders, tracks and sectors. Archaic things that are not as relevant as before (moving the head, seek latency). And anyway it's abstracted away.
    ii. The hardware abstracts this all away and give the OS/FS a picture a linear collection of sectors. That we can best read and written to in big linear chunks
    iii. Granularity of the device (512-byte block is the typical minimum read/write size)
    iv. Logical block numbers (maybe easier to think of this as "physical" block number)

Lecture #25, May 29, 2019

Mention Sector as the minimum read/write size

Logical block number (LBN), more like physical block number

4. Layout on storage Media (on-disk structures)
    i. Deep dive: FAT, do NTFS after Memory mapped files
        1. Sector 0, boot sector, contains the FAT identification, and info about the size of the drive
        2. The FAT, 12, 16 and 32 bit size fat
            a. Free clusters
            b. Cluster allocated to files and who they are chained together.
            c. Limited to 4GB
        3. The Root directory
        4. The Data area
        5. Dirents (32 bytes)
            a. File names (8.3 and long names), type, date (creation, Last Access, Last Write, size, first cluster number, protection)
        6. Creating a file
        7. Opening a file
        8. Writing to a file
        9. Deleting a file
        10. Subdirectories
        11. NT added a dirty bit to the Boot sector

Lecture #26, May 31, 2019

A word about fragmentation. Internal versus external on a HDD.

Today Solid State Drives (SSD) vs Hard Disk Drives (HDD)

5. Deep dive: persistent storage devices
    i. Hard drives
        1. The FS bends over backwards to try and keep data contiguous with minimal seek and transfer time.
        2. External Fragmentation is a BIG issue
    ii. SSD
        1. NAND based SSD
        2. Read operation
            a. Random read blocks, pretty fast
        3. Write operation
            a. Erasure blocks typically 128 to 512 pages
            b. Wear leveling
            c. Pretty slow, write Amplification
            d. Write Alignment for the FS to help with this
            e. TRIM ioctl
        4. Other Issues
            a. Over Provisioning
            b. Andrew Huang, www.bunniestudios.com look for "On Hacking MicroSD Cards"
        5. A different paradigm for the file system
            a. Less on fragmentation
    iii. RAID
        1. Raid 0 – nothing
        2. Raid 1 – mirroring
        3. Raid 5 – stripping with parity

Lecture #27, June 3, 2019

Finish RAID 5 analysis

1. Recovery

Word about buffer and non-buffered I/O

28. Back to MM and memory mapped files
    1. Deep dive: Memory-Mapped files
        i. write-through versus write-back
        ii. big part of Windows MM and Cache Manager

Lecture #28, June 5, 2019

Talk about logging and transactions at a fairly high level.

- Go through simple transaction
    - Two write areas, main data, and the log
    - Start Tx (prepare); write x to log, write y to log; commit Tx; (write-back) write x to main storage, write y to main storage, After copy back (end Tx)

Issues

- Handling simultaneous transactions
- Flushing to nonvolatile storage
- Idempotent

29. Deep dive: NTFS and File System Driver Implementation
    1. 2^64 = 16 exabytes
    2. Main components
        i. MFT
            1. File Records
            2. Resident and nonresident data
            3. Named data streams
            4. B+ Trees
        ii. Bitmap for cluster allocation and MFT record allocation
        iii. Logging file system actions
    3. The idea is to represent a small view of what is on disk in memory. In terms of files, directories, and things like bitmaps.
    4. In windows all of this is done via mapped files
    5. Delayed closing of files
    6. Tunneling of file names and other information, what it is used for
    7. Caching of files data
    8. Spare and compressed files

Lecture #29, June 7, 2019

Final is closed book.

Class recap and answer any questions.

Back to Scheduling

1. Multiprocessor versus uniprocessor issues

   Start with a simple UP system then build up to a MP system and add caches its behavior changes. Use two separate processes and then talk about how threads in the same process might best utilize a MP system.

   i. Who added caches anyway?

   ii. Processor affinity

   Talk about the required fixes the scheduler to handle MP systems and why

   Cores versus logical processors

2. Thrashing – too many tasks causing too many context switches.

   iii. Draw the curve and have them explain its behavior

3. Recap: FIFO, SJF and Shortest Remaining Job, Round Robin, Priority Queues, MLFQ

30. (Not in Textbook) A final look at the overall OS
    1. The reentrant kernel i.e., check previous mode.
    2. Deadlocks and recursion in the kernel
31. (3) Odd and Ends
    1. (Chapter 2.10) Virtual Machines
    2. Meltdown, Spector and other attacks