Concurrency

1

1

Why employ concurrency?

- Resource sharing, information exchange, collaboration
- Tolerate delays such as slow I/O devices
- Provide good response times, e.g., with human interaction
- Separate logical tasks
  - Garbage collection
  - Separate logical flow for each client in a concurrent server
- Reduce latency by deferring work
- Execute in parallel on hardware such as multicore machines

2

2

Concurrent Programming is Hard!

- The human mind tends to be sequential
- The notion of time is often misleading
- Thinking about all possible sequences of events in a computer system is at least error prone and frequently impossible

3

3

What do we need?

- Communication
  - Messages versus shared memory
- Coordinate
  - Synchronization
    - Mutual exclusion
    - Events

4

4

Why is Parallel Computing Hard?

- Amdahl's law – insufficient available parallelism
  - Speedup = 1/(fraction_enhanced/speedup + (1-fraction_enhanced)
- Overhead and complexity of communication and coordination
  - Classic problems concurrent programs
    - ***Races:*** outcome depends on arbitrary scheduling decisions elsewhere in the system
    - ***Deadlock:*** improper resource allocation prevents forward progress
    - ***Livelock / Starvation / Fairness***: external events and/or system scheduling decisions can prevent sub-task progress
- Portability – knowledge of underlying architecture often required

5

5

Steps in the Parallelization Process

- Decomposition into tasks
- Assignment to processes
- Orchestration – communication of data, synchronization among processes

6

6

One Strategy to Sum Vector Elements in Parallel (pseudo-code)

```
#define NUMPROCS 4
#define LENGTH 1024
int array[LENGTH]; /* initialized elsewhere */
int sum = 0;

int combine(pid)
{
    long int i, sum = 0;
    for (i = pid; i < LENGTH; i += NUMPROCS) {
       sum = sum + array[i];
    }
    return sum;
}
```

Question: Assuming a direct-mapped cache size of 4 Kbytes and line size of 16 bytes, how many misses will each process incur on "array" assuming they are running in parallel on separate processors/cores?

7

7

Types of Dependences

- Flow (or True) dependence – RAW
- Anti-dependence – WAR
- Output dependence – WAW

8

8

Synchronization

- Basic types
  - Mutual exclusion
  - Events
- Components of a synchronization operation
  - Acquire method (enter critical section, proceed past event)
  - Waiting algorithm (busy waiting, blocking)
  - Release method (enable others to proceed)

9

9

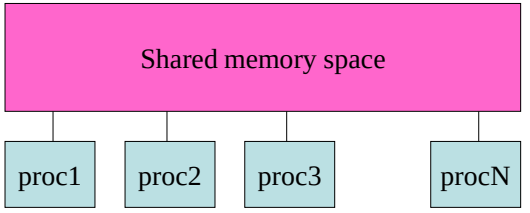
Data Sharing: CPU and Cache Support

- Special atomic read-modify-write instructions
  - Test-and-set, fetch-and-increment, load-linked/store conditional
- Coherent caches
  - Ensure that modifications propagate to copies

10

10

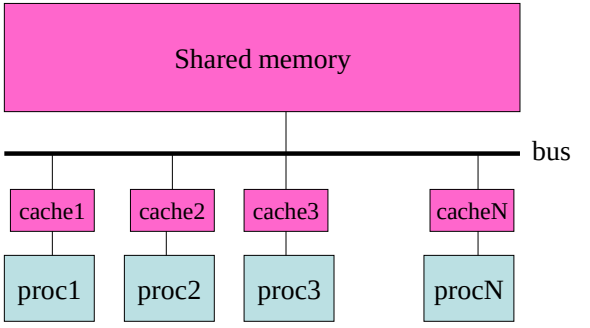
Shared Memory: A Look Underneath



11

11

Physical Implementation



12

12

Snoop-Based Coherence

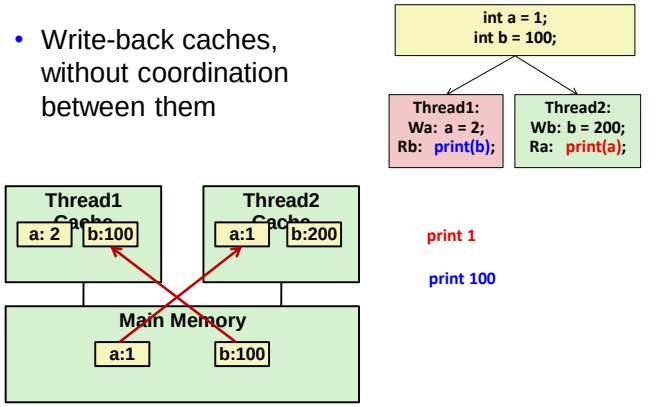
- Makes use of a shared broadcast medium to serialize events (all transactions visible to all controllers and in the same order)
  - Write update-based protocol
  - Write invalidate-based (e.g., basic MSI, MESI protocols)
- Cache controller uses a finite state machine (FSM) with a handful of stable states to track the status of each cache line
- Consists of a distributed algorithm represented by a collection of cooperating FSMs

13

13

Non-Coherent Cache Scenario

- Write-back caches, without coordination between them

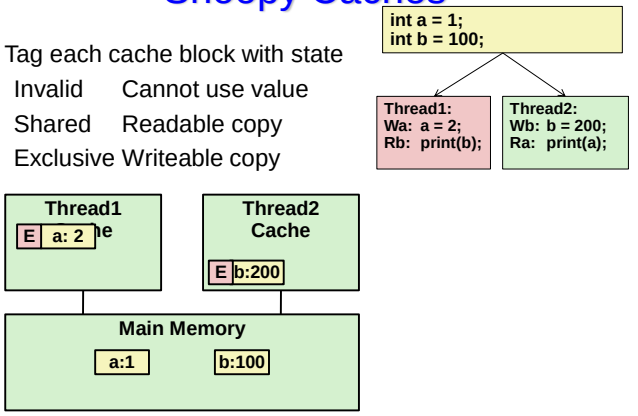


14

14

Snoopy Caches

- Tag each cache block with state
  - Invalid Cannot use value
  - Shared Readable copy
  - Exclusive Writeable copy

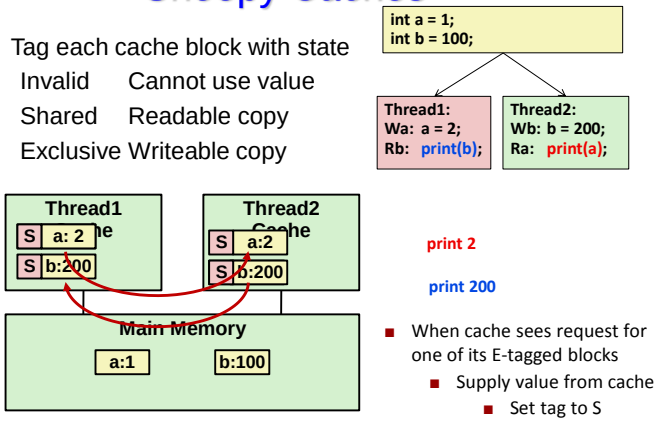


15

15

Snoopy Caches

- Tag each cache block with state
  - Invalid Cannot use value
  - Shared Readable copy
  - Exclusive Writeable copy



- When cache sees request for one of its E-tagged blocks
  - Supply value from cache
    - Set tag to S

16

16

Lessons Learned

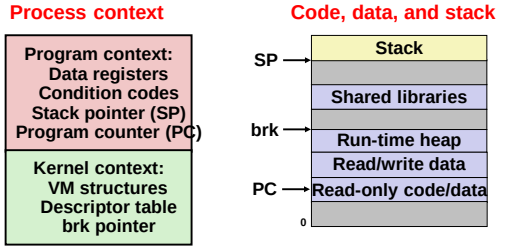
- Must have parallelization strategy
  - Partition into K independent parts
  - Divide-and-conquer
- Inner loops must be synchronization free
  - Synchronization operations very expensive
- Beware of Amdahl's Law
  - Serial code can become bottleneck
- You can do it!
  - Achieving modest levels of parallelism is not difficult
  - Set up experimental framework and test multiple strategies

17

17

Traditional View of a Process

- Process = process context + code, data, and stack

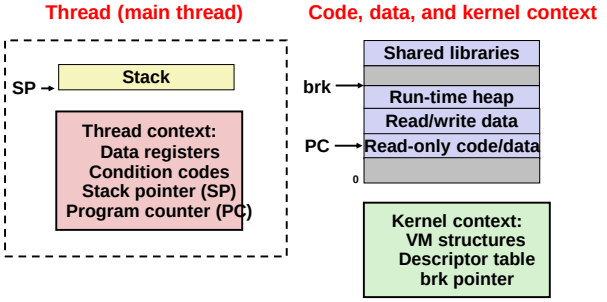


20

20

Alternate View of a Process

- Process = thread + code, data, and kernel context

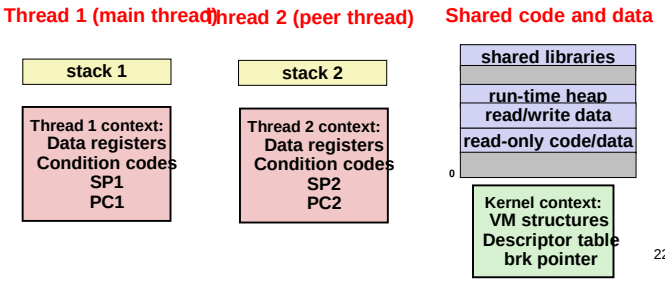


21

21

A Process With Multiple Threads

- Multiple threads can be associated with a process
  - Each thread has its own logical control flow
  - Each thread shares the same code, data, and kernel context
  - Each thread has its own stack for local variables
    - but not protected from other threads
  - Each thread has its own thread id (TID)



22

22

Threads vs. Processes

- How threads and processes are similar
  - Each has its own logical control flow
  - Each can run concurrently with others (possibly on different cores)
  - Each is context switched
- How threads and processes are different
  - Threads share all code and data (except local stacks)
    - Processes (typically) do not
  - Threads are somewhat less expensive than processes
    - Process control (creating and reaping) twice as expensive as thread control
    - Linux numbers:
      - ~20K cycles to create and reap a process
      - ~10K cycles (or less) to create and reap a thread

Wrap-Up

Computer Organization

- Goal: Understanding of the inner workings of modern computer systems
- Study the hierarchy of layers that comprise computer systems
  - Hardware
  - Systems software
  - Applications software

Topics covered:

- Data representation and computer arithmetic
- Assembly-level programs and instruction-set architectures
- Processor architectures
- Memory and storage hierarchies
- Performance optimization
- Exceptional control flow
- Processes and virtual memory
- I/O – interface, storage technologies, networking
- Concurrency

Number Representation

An n digit number can be represented in any base as

MSD … LSD

n-1 … 0

The value of the *i*th digit d is d x base^{i}, where *i* starts at 0 and increases from right to left

Decimal (base 10) is the natural human representation, binary (base 2) is the natural computer representation

E.g. $1100_2 = 1x2^3 + 1x2^2 + 0x2^1 + 0x2^0 = 12_{10}$

30

Data Representation

- Memory: a large single-dimensional, conventionally byte-addressable, untyped array
- Byte ordering – big versus little endian
- Possible common interpretations
  - Instruction
  - Integer
  - Floating point
  - character

31

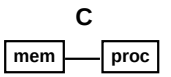
Assembly Characteristics

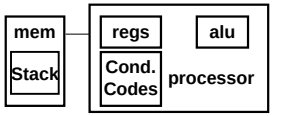
- Minimal Data Types
  - "Integer" data of 1, 2, or 4 bytes
    - Addresses (untyped pointers)
    - Data values
  - Floating point data of 4, 8, or 10 bytes
  - No aggregate types such as arrays or structures
    - Just contiguously allocated bytes in memory
- Primitive Operations
  - Perform arithmetic function on register or memory data
  - Transfer data between memory and register
    - Load data from memory into register
    - Store register data into memory
  - Transfer control
    - Unconditional jumps to/from procedures
    - Conditional branches

32

Summary: Abstract Machines

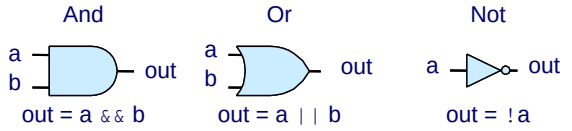
Machine Models	Data	Control
C	1) `char` 2) `int`, `float` 3) `double` 4) `struct`, array 5) pointer	1) loops 2) conditionals 3) `switch` 4) Proc. call 5) Proc. return
Assembly	1) byte 2) 2-byte word 3) 4-byte long word 4) contiguous byte allocation 5) address of initial byte	3) branch/jump 4) `call` 5) `ret`



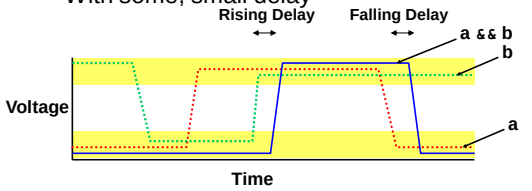


33

Computing with Logic Gates

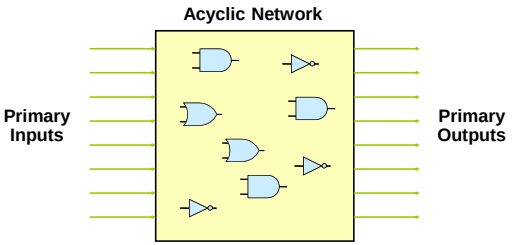


- Outputs are Boolean functions of inputs
- Respond continuously to changes in inputs
  - With some, small delay



34

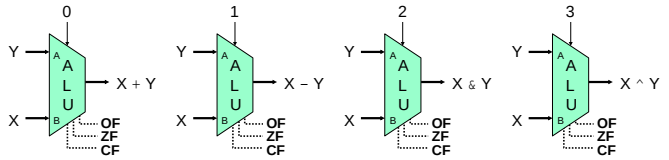
Combinational Circuits



- Acyclic Network of Logic Gates
  - Continously responds to changes on primary inputs
  - Primary outputs become (after some delay) Boolean functions of primary inputs

35

Arithmetic Logic Unit



- Combinational logic
  - Continuously responding to inputs
- Control signal selects function computed
  - Corresponding to 4 arithmetic/logical operations in Y86
- Also computes values for condition codes

36

Sequential Logic: Memory and Control

- Sequential:
  - Output depends on the ***current*** input values and the ***previous*** sequence of input values.
  - Are Cyclic:
    - Output of a gate feeds its input at some future time.
  - Memory:
    - Remember results of previous operations
    - Use them as inputs.
  - Example of use:
    - Build registers and memory units.

37

Clocks

- Signal used to synchronize activity in a processor
- Every operation must be completed in the time between two clock pulses (or rising edges) --- the cycle time
- Maximum clock rate (frequency) determined by the slowest logic path in the circuit (the critical path)

38

Processor Summary

- Design Technique
  - Create uniform framework for all instructions
    - Want to share hardware among instructions
  - Connect standard logic blocks with bits of control logic
- Operation
  - State held in memories and clocked registers
  - Computation done by combinational logic
  - Clocking of registers/memories sufficient to control overall behavior
- Enhancing Performance
  - Pipelining increases throughput and improves resource utilization
  - Must make sure maintains ISA behavior

39

Code Optimization Summary

- Loop fusion and unrolling
- Code Motion
  - ***Compilers are good at this for simple loop/array structures***
  - ***Don't do well in presence of procedure calls and memory aliasing***
- Reduction in Strength
  - Shift, add instead of multiply or divide
    - *compilers are (generally) good at this*
    - *Exact trade-offs machine-dependent*
  - Keep data in registers rather than memory
    - *compilers are not good at this, since concerned with aliasing*
- Share Common Subexpressions
  - ***compilers have limited algebraic reasoning capabilities***

40

Locality

- Principle of Locality:
  - Programs tend to reuse data and instructions near those they have used recently, or that were recently referenced themselves.
  - Temporal locality: Recently referenced items are likely to be referenced in the near future.
  - Spatial locality: Items with nearby addresses tend to be referenced close together in time.

Locality Example:

```
sum = 0;
for (i = 0; i < n; i++)
    sum += a[i];
return sum;
```

- **Data**
  - Reference array elements in succession (stride-1 reference pattern): **Spatial locality**
  - Reference `sum` each iteration: **Temporal locality**
- **Instructions**
  - Reference instructions in sequence: **Spatial locality**
  - Cycle through loop repeatedly: **Temporal locality**

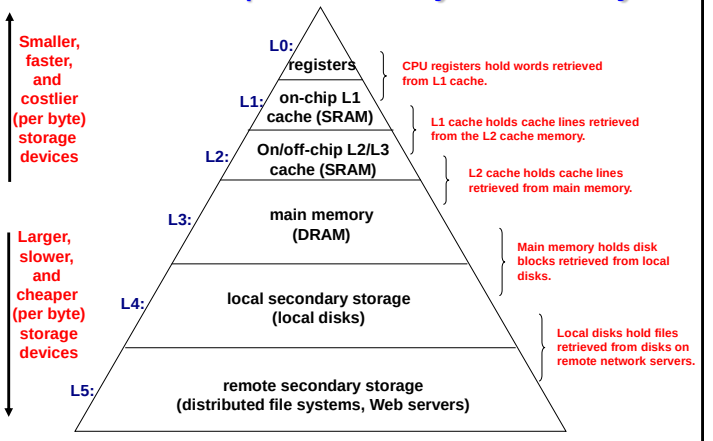
41

Memory Hierarchies

- Some enduring properties of hardware and software:
  - Fast storage technologies cost more per byte and have less capacity
  - The gap between CPU and main memory speed is widening
  - Well-written programs tend to exhibit good locality
- These fundamental properties complement each other beautifully
- They suggest an approach for organizing memory and storage systems known as a memory hierarchy

42

An Example Memory Hierarchy



43

Caches

- Cache: A smaller, faster storage device that acts as a staging area for a subset of the data in a larger, slower device.
- Fundamental idea of a memory hierarchy:
  - For each k, the faster, smaller device at level k serves as a cache for the larger, slower device at level k+1.
- Why do memory hierarchies work?
  - Programs tend to access the data at level k more often than they access the data at level k+1.
  - Thus, the storage at level k+1 can be slower, and thus larger and cheaper per bit.
  - Net effect: A large pool of memory that costs as much as the cheap storage near the bottom, but that serves data to programs at the rate of the fast storage near the top.

44

Cache Organization

- Location/placement – how do you locate a block/where is a block placed
- Replacement – which block do you replace
  - Least recently used (LRU)
  - FIFO
  - Random
- Write policy – what happens on a write
  - Write back
  - Write through no write allocate
  - Write through write allocate

45

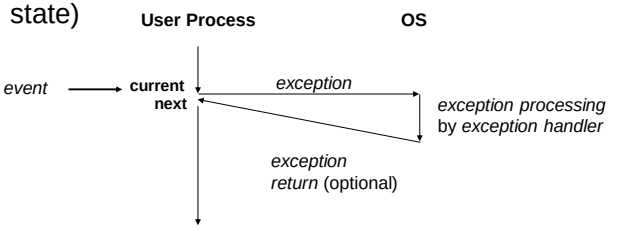
Problem 1

- Consider a computer with a 12-bit address space (i.e., each memory address is 12-bit long) and a byte-addressable memory. It has a data cache capable of holding eight cache blocks. Each cache block is 4 bytes (excluding any overhead).
  - How many tag bits are needed
  - What is the content of the cache for the following access pattern that repeats 4 times (a loop)
    - 0x200, 0x204, 0x208, 0x20C, 0x2F4, 0x2F0, 0x200, 0x204, 0x218, 0x21C, 0x24C, 0x2F4

46

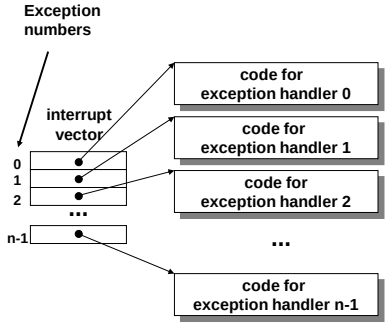
Exceptions

- An *exception* is a transfer of control to the OS in response to some *event* (i.e., change in processor state)



47

Interrupt Vectors



- Each type of event has a unique exception number k
- Index into jump table (a.k.a., interrupt vector)
- Jump table entry k points to a function (exception handler).
- Handler k is called each time exception k occurs.

48

Asynchronous Exceptions (Interrupts)

- Caused by events external to the processor
  - Indicated by setting the processor's interrupt pin
  - handler returns to "next" instruction.
- Examples:
  - I/O interrupts
    - hitting ctl-c at the keyboard
    - arrival of a packet from a network
    - arrival of a data sector from a disk
  - Hard reset interrupt
    - hitting the reset button
  - Soft reset interrupt
    - hitting ctl-alt-delete on a PC

49

Synchronous Exceptions

- Caused by events that occur as a result of executing an instruction:
  - Traps
    - Intentional
    - Examples: system calls, breakpoint traps, special instructions
    - Returns control to "next" instruction
  - Faults
    - Unintentional but possibly recoverable
    - Examples: page faults (recoverable), protection faults (unrecoverable).
    - Either re-executes faulting ("current") instruction or aborts.
  - Aborts
    - unintentional and unrecoverable
    - Examples: parity error, machine check.
    - Aborts current program

50

Exceptional Control Flow

  - Mechanisms for exceptional control flow exists at all levels of a computer system
- Low level Mechanism
  - exceptions
    - change in control flow in response to a system event (i.e., change in system state)
  - Combination of hardware and OS software
- Higher Level Mechanisms
  - Process context switch
  - Signals
  - Nonlocal jumps (setjmp/longjmp)
  - Implemented by either:
    - OS software (context switch and signals)
    - C language runtime library: nonlocal jumps

51

Processes

- Def: A *process* is an instance of a running program.
  - One of the most profound ideas in computer science.
  - Not the same as "program" or "processor"
- Process provides each program with two key abstractions:
  - Logical control flow
    - Each program seems to have exclusive use of the CPU.
  - Private address space
    - Each program seems to have exclusive use of main memory.
- How are these Illusions maintained?
  - Process executions interleaved (multitasking)
  - Address spaces managed by virtual memory system

52

Virtual Memory

- Programmer's View
  - Large "flat" address space
    - Can allocate large blocks of contiguous addresses
  - Processor "owns" machine
    - Has private address space
    - Unaffected by behavior of other processes
- System View
  - User virtual address space created by mapping to set of pages
    - Need not be contiguous
    - Allocated dynamically
    - Enforce protection during address translation
  - OS manages many processes simultaneously
    - Continually switching among processes
    - Especially when one must wait for resource
      - E.g., disk I/O to handle page fault

53

Problem 2

An ISA supports an 8-bit, byte-addressable virtual address space. The corresponding physical memory has only 128 bytes. Each page contains 16 bytes. So there are 8 pages in the physical memory. A simple, one-level translation scheme is used and the page table resides in physical memory. The initial contents of the physical memory are shown below.

Physical Page Number	Physical Page Contents
0	Empty
1	Virtual Page 13
2	Virtual Page 5
3	Virtual Page 2
4	Empty
5	Virtual Page 0
6	Empty
7	Page Table

A three-entry Translation Lookaside Buffer that uses LRU replacement is added to this system. (Note: LRU policy is used to select virtual pages for replacement in physical memory) Initially, this TLB contains the entries for virtual pages 0, 2, and 13. Assume this machine executes a program that issues the following references to the virtual memory:
References (to virtual pages): 0, 13, 5, 2, 14, 14, 13, 6, 6, 13, 15, 14, 15, 13, 4, 3.
1.List the references that generate a TLB hit
2.List the references that generate a page fault.
3.What is the hit rate of the TLB for this sequence of references?
4.At the end of this sequence, what three entries are contained in the TLB?
5.At the end of this sequence, what are the contents of the 8 physical frames?

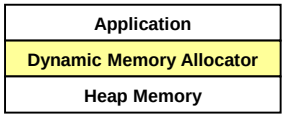
54

Harsh Reality: Memory Matters

- Memory is not unbounded
  - It must be allocated and managed
  - Many applications are memory dominated
    - Especially those based on complex, graph algorithms
- Memory referencing bugs especially pernicious
  - Effects are distant in both time and space
- Memory performance is not uniform
  - Cache and virtual memory effects can greatly affect program performance
  - Adapting program to characteristics of memory system can lead to major speed improvements

55

Dynamic Memory Allocation



- Explicit vs. Implicit Memory Allocator
  - Explicit: application allocates and frees space
    - E.g., `malloc` and `free` in C
  - Implicit: application allocates, but does not free space
    - E.g. garbage collection in Java, ML or Lisp
- Allocation
  - In both cases the memory allocator provides an abstraction of memory as a set of blocks
  - Doles out free memory blocks to application

56

Memory-Related Bugs

- Dereferencing bad pointers
- Reading uninitialized memory
- Overwriting memory
- Referencing nonexistent variables
- Freeing blocks multiple times
- Referencing freed blocks
- Failing to free blocks

57

Topics covered:

- Data representation and computer arithmetic
- Assembly-level programs and instruction-set architectures
- Processor architectures
- Memory and storage hierarchies
- Performance optimization
- Exceptional control flow
- Processes and virtual memory
- I/O – interface, storage technologies, networking
- Concurrency

58

58

THANK YOU!

Wishing you a restful, safe, and warm holiday season, and a Happy New Year!

59