a) (RISC) Reduced Instruction Set Computing - reduces instruction set for quicker processing by using simple instructions but more of them
b) (CISC) Complex Instruction Set Computing - full instruction set, provides all needed capabilities. - complete task in as few lines of assembly as possible
c) (Vector Instructions) SSE provided help with floating point instruction types while MMX did integer. SSE providing integer instructions in a later release.
d) (Condition Codes) Requires less compare and test instructions when converting to assembly, but provides extra complexity to both the programmer and the compiler
e)(Stack) Data type that is easy to store large quantities of data in. LIFO - last in first out, can only access most recently entered data
f) (iRAM) Intelligent RAM - tried to find a better tradeoff between cost and performance by integrating DRAM on the chip and eliminating the need for a redundant static memory cache.

2. (argue against > 8 registers) Performance improvement would be comparable to a logarithmic function, so performance improvements would be minor in most situations, while cost would increase drastically, as well as space required.

3. (what does linker do) The linker takes the code from the compiler, and replaces variables with specific memory addresses in order to create an executable.

4. (movl 8(%ebp), %edx) copies value of ebp + offset 8 into edx

5. (what is SetX instructions used for) Sets environment variables for the currently logged on user or the machine.

6. (why goto/jmp considered poor programming) Goto/jmp statements are dangerous to use in code because of how easily they can cause unexpected issues and the difficulty of debugging such an issue.

7. (translating loops to do-while change chip architecture)

8. (unconditional branch incur no overhead but conditional do) An unconditional branch always executes if it is reached, while a conditional branch has to check a set of parameters before execution.

9. (2 ways switch statements is improved by compilers) If the switch has a dense range of case expressions, then the switch is turned into a jump table which is very quick and easy to execute. If its impossible to use a jump-table then the compiler will reorder the comparisons in order to use a binary search.

10. (AMD-64 vs Itanium debate) AMD retained all backwards compatibility while providing specifications for 64 bit machines. Itanium is based on instruction-level parallelism which made Itanium very inefficient with old x86 applications. Due to the lack of backwards compatibility, AMD made leaps in the computing market while Itanium ended up with a very specific userbase (In 2009, HP had over 95%of the server market share).

a. Pipelining - when seperate phases of code are executing simulatenously to improve performance
b. %rdi - used for integer and memory address arguments
c. %esp - register that will always point to the top of the stack
d. Stack frame - space allocated on the stack, holds neccesary information to save and restore the state of a procedure
e. Address alignment - keeps processor from grabbing only half of a word so not to than wasting its clock cycle
f. Caller save / callee save - caller save means the current stack frame will save register values, callee means the stack frame for the function being called will save the registers. callee is generally faster because it will only save the registers it uses rather then all of the registers.

2. Why use the lea instruction for mathematical operations when it is designed for calculating memory addresses? What limitations does it have?: ability to perform addition with either 2 or 3 operands and ability to store the result in any register. - doesnt affect condition flags

3. How are 1D & 2D arrays represented in memory? : stored linearly

4. How is nested array element access performed?

5. How is multi-level array element access performed?

6. What is assert() and why do we use it? : procedure that declares 'this(w/e your looking for)' has to be true or it will crash.

7. What happens in a procedure call? : old register values are stored in memory, procedure stored at top stack and executed, when procedure finished old register values restored and poped off stack

8. How does the stack make recursion work? : the stack adds recursive calls to top of stack and pops off when no longer needed

9. Describe what 8(%ebp) is using normal compiler conventions in IA32. : holds local variables

10. Why is x86-64's passing of arguments to procedures significantly faster than IA32? : 64 has 2x amount of register can hold more values

a. caching - *storing data in cache memory. Quick accessibility*
b. locality - *when same value or related storage locations accessed frequently.*
c. DMA – Direct Memory Access – *allows certain hardware to access system memory independent from the cpu*
d. Conflict miss - failed attempt to read/write data in the cache – *could have been avoided had the cache not evicted an earlier entry.*
e. Capacity miss – *misses occur due to the finite size of the cache. Because its always full new entrys will overwrite older ones.*
f. Direct-mapped cache – *when each block from main memory has only one place it can appear in the cache. Allows simple and fast speculative execution.*
g. Set-associative cache - *between direct-mapped and fully associative caches. Each line is divided into sets and the middle bits of the address determine the set it is placed.*
h. Write-through caching - *disk or memory cache that supports caching or writing. After writing if a read is called, the read performance is improved because its already in high-speed cache.*

2. On a modern CPU, if you had the choice of spending 100K instructions or waiting for one disk seek to complete (assuming an average 7200RPM 3.5" drive), which would you choose?
100K instructions are still faster then then one disk seek.

3. Briefly describe the process of reading a bit from DRAM.
Requires reading a row from the memory array into the row buffer and writing it back unchanged.

4. What is the net effect of the cache hierarchy?
The cache hierarchy is a way for multiple cores to have shared cache.

5. Why would register usage be controlled by the compiler, but cache usage controlled by the CPU hardware?
Because not all variables are in use the compiler must decide how to allocate these variablaes to a small, finite set/not all processors will share access to the same memory.

6. Assume you are specifying the hardware for a program randomly accessing a large amount (think much larger than a computer's average RAM size). Which would be a better way to spend your budget – more cache, or more RAM?
Ram would have a larger impact on performance because cache sizes are small.

7. Discuss the relative merits of write-back vs. write-through caching.
Write through – data written to cache and disk (large write time dropping performance)
write back – data is only written to cache until cache is full, after full it'll write to HD
cache is power sensitive, so if power is lost the data in cache is gone.

8. Why did the blocked array multiply exhibit better caching performance?
Block multiplication reuses more data then regular array multiplication. This allows cache to be used more effectively

9. Discuss your decision process when choosing between a faster dual-core chip (with half the cache of the quad-core) or a slower quad-core chip for a given application.
Dealing with a small data size cache speeds it up more. So quad core works better.
Larger data sets dependent on processor speed

10. Applications are usually classed as CPU-bound or I/O-bound. How would you figure out which class to put a particular application end?
Cpu-bound – run themselves with little to no input from user
IO-bound – dependent on user input, which is primary cause for slowness

static linking - linker copying all library routines used in the program into the .exe image - faster/portable but requires more memory/disk space

Dynamic linking - places name of a sharable library in .exe img - multiple programs can share a single copy of the library

Relocation - process of assigning load addresses to various parts of a program

strong/weak symbols - functions/initilized global variables = strong; unintialized global vars get weak

virtual memory - technique that virtualizes various forms of computer storage (RAM/disk storage) allowing a program to be desgined as though there is only one kind of memory, behaves like normal ram

sandbox – security mechanism, seperates running programs. Used to execute untested or untrusted code from unverified suppliers, users, websites etc..

2. Compare distributing a statically-linked application to distributing a VM image (such as one for VMWare) for a commercial software firm.

3. How does operator/method overloading complicate a linker's job? How do they work with compilers to resolve these issues?

4. How does a DLL/.so file differ from a typical .o file? A '.o' is an object file where as a DLL/.so are static libraries

5. Why would the ability to link a DLL into a running application be useful? Give two examples.

DLL can be shared between multiple processes, if a bug is found in the DLL you can just rebuild the DLL rather then the whole program.

6. Why would "library interpositioning" be useful for both security professionals and hackers?

Will allow a security professional to preload a specific set of security functions when they open a program. Likewise a hacker could just as easily preload malicious code

7. Would static or dynamic inking tend to offer better locality on a typically PC with virtual memory enabled?

Dynamic linking – reduces overall space required for a program thus offering more locality

a. Signal - voltage that represents coded information
b. Interrupt - signal that stops the main function and figures out what to do next
c. Interrupt vector – list comprising locations of various interrupt handlers
d. Trap - aka exception – interrupt caused by an exceptional condition (breakpoint, division by zero etc..)
e. Process - instance of a computer program
f. Context switch - procedure the cpu follows to switch from 1 task to another, neither conflict with e/o
g. Zombie – computer connected to the internet that has been compromised by a hacker.
h. Exec() - similar to fork – starts a new process
i. Quantum - individual unit of data

2. If an I/O-bound application typically exhibits a low degree of locality, what effect is this likely to have on its overall performance?

Little to no decrease because programs waiting for the cpu

3. Why would modern caches index virtual rather than physical addresses?

Virtual address space doesn't need to be physically contiguous

4. Give an example of a typical fault arising from a memory reference. Is it necessarily a bad thing?

you can force a page fault to cause a program to assign more virtual memory to a program

5. What is the difference between a process and a thread?

Processes are completely independent of each other unless the OS specifically handles interaction, threads share the same chunk of memory and communicate directly.

6. When fork() returns, how do you know whether you are the child or parent process?

Childs pid = 0; Parents pid = positive non zero #

7. Does an average single-threaded user program need to worry about concurrency? Why or why not?

Concurrency handles a system that has multiple processes using the same piece of data, so in a single threaded machine there is no need for concurrency.

8. Why arrange the processes in a hierarchy? Why not just make them all “equal”?

Easier for the user the understand a hierarchy when you need to set priorities

9. Why would the default action upon receiving a signal be program termination?

Program termination is the safest way to handle an unexpected signal

10. Evaluate using signals rather than global variables to control program behavior.

Signals are expensive, so unless your running a large program global variables will keep performance from deterring.

a. Page table - where the operating system stores its mappings of virtual addresses to physical addresses

b. TLB - a cache exclusively for the page table which allows the page table to look up the physical addresses at a cache level speed

c. Multi-level page table – page table of page tables, first table is broad linking to other tables that cover all available memory.

d. Memory mapping a file - a file, we can greatly increase I/O performance, especially on large files. It is however possible that by memory-mapping a file, we create more page faults, which could actually slow down performance.

e. Demand paging - A disk page only gets copied into physical memory if an attempt is made to access it. This can cause a very large number of page faults if memory is rarely re-used, but will save a lot of disk lookups if memory is reused constantly.

2. How does virtual memory simplify memory management for the compiler?

   Virtual memory allows the compiler to always start at memory address 0, and makes finding "available" chunks of contiguous memory very easy, since it doesn't have to be continuous in physical memory.

3. How does virtual memory protect address spaces from each other?

   By using paged virtual memory, it makes it impossible for a program to access a page that is not allocated to that program. Attempting to do so would cause a page fault.

4. How does the OS handle a page fault?

   The operating system tries to handle the page fault by making the required page accessible at a location in physical memory or kills the program in the case of an illegal access.

5. How does virtual memory allow multiple programs to share the same memory?

   When using virtual memory we can have multiple virtual addresses that all point to the same chunk of virtual memory (read-only access) allowing the memory to be shared.

6. How do multi-level page tables reduce the overhead due to virtual memory?

   A normal page table has to have an entry for every single memory address, whether its being used or not. A multi-level page table is essentially a page table of page tables. The first page table has very broad links to other page tables that cover all available memory. The second layer of page tables only have entries for addresses that are actually used. For your average program that doesn't use all or most of the available memory, this causes a large speedup. However if a program is using all available memory and thrashing then this approach will cause an even larger drop in performance.

7. What does the valid bit indicate in a TLB entry?

   The valid bit of a TLB says whther the entry is currently usable. If the valid bit is 0, the computer must go out to disk to find the page.

8. How does memory-mapped file I/O differ from normal I/O from a programmer's perspective?

   CPU instructions that can access memory can also be used for accessing devices because memory-mapping uses the same address bus.

9. Give an example where demand paging could be less efficient than normal I/O?

   If you will be using most or all of the pages, then demand paging will cause several page fault traps, which cause a program to slow down immensely

.

10. How does memory-mapping speed typical the Fork() operation?

    Memory mapping removes the need to copy the pages of the original parent by allowing the child to share the pages.

a. Exec() - completely replaces the current running process with program passed as an argument
b. Copy on Write – optimization strategy - fundamental idea is that if multiple callers ask for resources which are initially indistinguishable, they can all be given pointers to the same resource
c. Allocator - Allocators handle all the requests for allocation and deallocation of memory for a given container
d. Internal fragmentation – caused when payload is smaller than block size
e. External fragmentation - when there is enough aggregate memory but no block large enough

2. How would a context switch between applications affect the memory subsystem's performance?

By completely flushing the TLB, we can prevent TLB entries from becoming invalid (since virtual to physical mapping is different)

3. Why is "demand paging" generally more efficient than loading an entire application into memory?

Eventually it will keep the pages accessed the most, where as loading an entire application will result in many misses.

4. In our "fast file copy example", how does it avoid copying data into 'user space', and why does this speed things up?

5. Contrast and compare explicit vs. implicit memory allocation.

Explicit – application allocates & frees space (malloc and free)
Implicit – application allocates but does not free (garbage collection in java)

6. What affect would allowing allocators to reorder or buffer requests potentially have?

Pointers directed towards malloc'd space may no longer point to the correct location if the buffer is compacted

7. How would you implement the ability to move blocks once they've been allocated? (Hint – think early MacOS 'handles'.)

Mac OS supports a form of relocatable heap block, which is accessed indirectly via a handle which points into a *master pointer* block (a non-relocatable heap block). Heap blocks can optionally be relocatable, and relocatable blocks can be locked. Relocatable blocks are compacted from time to time to avoid external fragmentation. Relocatable blocks can also be marked purgeable, which means the system may free them during compaction if memory becomes low.

8. Does padding for byte alignment comprise internal or external fragmentation? Why?

Internal – bye padding leaves empty padding on each end.

9. Why can't malloc use the same techniques we use for virtual address spaces?

Because once we malloc something it is a fixed space we cant changes the allocated size

10. Compare and contrast immediate and deferred coalescing. How do these issues compare to Java/.Net garbage collection design issues?

Deferred coalescing is delayed joining of free blocks, immediate joins them as soon as they're free.
Since deferred waits until it has enough it will do them all at once, causing better performance however due to the stipulations of block management delayed coalescing is not useable in a real-time situation.

11. Based on the design discussed in the course notes, what is the limiting factor on the size of request you can make to malloc?

You're limited by how large of a contiguous block you can find.

a. Socket - he connector on the motherboard that houses a CPU and forms the electrical interface and contact with the CPU.

b. Frame – structure used to divide knowledge into substructures

c. DNS - naming system for computers, services, or any resource connected to the internet/private network

d. Packet sniffing – program or piece of hardware that can intercept and log traffic passing over a digital network.

e. HTTP – Hypertext Transfer Protocol – application protocol for distributed, collaborative hypermedia information systems, foundation of data communication for the world wide web.

f. MIME types - multi-purpose Internet Mail Extensions – form a standard way of classifying file types on the internet,so they can transfer files of the same type in the same way.

2. Why would more and more content on the WWW be dynamic rather than static? Briefly discuss the tradeoffs involved.

Static – must republish when changes are made or viewers will not see updates

Dynamic – you don't have to immediately republish when updates are made for the viewer to see changes

3. What two things must an internet protocol do?

Provide naming scheme – defines format for host addresses

Provide delivery mechanism – defines standard transfer unit (packet – consists of header & payload)

4. Which would be a more appropriate choice, typically, for implementing a VOIP solution – UDP or TCP?

Because we're concerned more about keeping the stream of information going rather than the order it comes in UDP is better for voip. TCP guarantess delivery of data

5. Why does designating a service as a socket pair (`IP_Address:Port_Number`) limit scalability and reliability?

6. What would you expect to be the bottleneck for a web server primarily serving out lots of static HTML documents that "feels slow/sluggish:" (a) another CPU, (b) faster storage, or (c) a faster Internet connection?

CPU – because most of the time spent processing the html is done using the cpu (parsing & executing the html document)

7. What would probably the most effective way to speed up a web server which does many random read-only lookups into a 3GB database, returning 1KB-2KB of HTML as the result for each lookup, assuming the Internet connection is nowhere near saturated?

Combine pages to make less lookups

8. Why would you want to make an I/O bound WWW server multithreaded on a single-core system?

By making the serve multithreaded, you allow it do other tasks while its waiting on I/O to be processed.

9. Why is MAC-address-based authentication such a bad idea?

MAC addressed too easy to spoof

10. How can a proxy located on the KSU campus accelerate web access to services provided by web sites like YouTube or Sports Illustrated?

Proxy's typically retain information in their cache longer (and hold more) than a normal web browser, resulting in dramatic performance increases for popular websites.

a. mutex - Mutexes happens to critical sections of code where only one process can touch at a time.

b. critical section – a piece of code that accesses a shared resource that must not be accessed by more than one thread of execution

c. semaphore - var that provides useful abstraction for controlling access by multiple processes to a common resource.

d. deadlock – situation which two or more competing actions are each waiting for the other to finish.

e. starvation – situation where a low priority action may never be executed.

f. thread-safe – code is thread-safe if it functions correctly during simultaneous execution by multiple threads.

g. race condition – when system attempts two or more operations at same time but needs them to be done in a sequential order.

2. Why would you use process-based (rather than thread-based) parallelization for building a web system? Give two examples.

1) Because processes are independent they contain their own state info, address spaces. This keeps each call for different users separate. 2) Threads typically do not need to be performed in a linear manner, this could be bad when dealing with user requests from a web server.

3. Is worrying about "race conditions" necessary in your average single-threaded application?

No, because in a single threaded application everything is done sequentially, you wont need to worry about multiple operations done at the same time.

4. Why do modern OS not worry too much about deadlock?

Because the OS preempts the CPU eliminating one of the four preconditions for deadlock.

5. What is an "atomic" operation? Why are these necessary?

Operation which a processor can simultaneously read a location and write it in the same bus. This prevents any other processor from writing or reading memory until it is complete.

6. Why would we want to keep critical sections as short as possible?

To avoid long delays for servicing other interrupts

7. How could you prevent deadlock in the "Dining Philosophers" problem?

If we have a mediator which all philosophers must request permission, he can oversee how many forks are given out and make the other wait until the previous is done.

8. Give a simple way to keep starvation from being an issue in a multithreaded system.

Implement a Fair Lock which utilizes a queue, allowing the top of queue to use the CPU, and the back waits