Operating Systems

1. Describe inheritance, multithreading, watchdog timer, etc.
    a. Inheritance: Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application. Inheritance allows to reuse the code functionally and allows fast implementation time.
    b. Multithreading: Multithreading is the ability of a program to manage its use by more than one user at a time and to even manage multiple requests by the same user without having to have multiple copies of the programming running in the computer.
    c. Watchdog timer: WDT is a piece of hardware that can be used to automatically detect software anomalies and reset the processor if any occur.
2. What is virtual memory and caches(Also read upon Cache coherency)
    a. Physical memory is scarce in machines, so virtual memory is an optimization to expand that
    b. Memory stored in pages in hard disk and by using caches
3. What are priority inversion, reentrancy, spinlocks
    a. Priority Inversion: lower priority task running with semaphore, higher priority is waiting for semaphore. Medium priority task interrupts, runs, and then high priority can run
        i. Solution is to bump priority of low priority to highest priority and then it can't be preempted by medium (It's called priority inheritance)
    b. Reentrancy: function is reentrant when during its execution if it gets interrupted, it can return and resume function without any change
        i. Can't use global/static data
    c. Spinlock: a task is waiting for a certain resource that is locked and is continually checking if it opened up, using most of the CPU resources
        i. Good rule of thumb - avoid them in user code
4. What is atomic programming/non-locking operation?
    a. Atomic operations are operations that are guaranteed to keep their values safe

    Atomicity

    In computer programming, an operation done by a computer is considered *atomic* if it is guaranteed to be isolated from other operations that may be happening at the same time. Put another way, atomic operations are indivisible.

5. What is concurrency, parallelism and multithreading?
    a. Concurrency is essentially applicable when we talk about minimum two tasks or more. When an application is capable of executing two tasks virtually at same time, we call it concurrent application. Though here tasks run looks like simultaneously, but essentially they MAY not. They take advantage of CPU time-slicing feature of operating system where each task run part of its task and then go to waiting state. When first task is in waiting state, CPU is assigned to second task to complete it's part of task. Operating system based on priority of tasks, thus, assigns CPU and other computing resources e.g. memory; turn by turn to all tasks and give them chance to complete. To end user, it seems that all tasks are running in parallel. This is called concurrency.

    b. Parallelism does not require two tasks to exist. It literally physically run parts of tasks OR multiple tasks, at the same time using multi-core infrastructure of CPU, by assigning one core to each task or sub-task. Parallelism requires hardware with multiple processing units, essentially. In single core CPU, you may get concurrency but NOT parallelism.
6. Difference between Thread and Process
    a. A process has all the resources needed to execute a program - memory space, registers, stack space, security etc
    b. A thread is a line of execution and exists within a process
7. Mutexes and semaphores? what is the main difference between them? what is the difference between binary semaphore and mutex? how does locking happen i ln mutex?
    a. Mutex is for exclusive access to a resource. A Binary semaphore should be used for Synchronization (i.e. "Hey Someone! This occurred!").
    b. Semaphores signal between tasks
        i. B gives semaphore and A takes it, a task doesn't give and take it by itself
    c. https://stackoverflow.com/questions/62814/difficult ference-between-binary-semaphore-and-mutex
8. What is thrashing? what is excessive paging? The main areas as seen on the believe as it is not
    a. Have virtual memory that is > RAM size
    b. So OS constantly is àswapping contents in and out of virtual memory
    c. Thrashing is when this uses most of the processing power
9. What is dynamic loading? what is static loading? when to use dynamic loading? what are the advantages? give an example when to use dynamic loading?
    a. Dynamic loading is when code is loaded into memory dynamically
    b. Static is when it's all resolved at compile time
10. Difference between regular OS and real-time OS
    a. RTOS used for time critical systems
    b. Task scheduling in RTOS is priority based, in regular OS is high throughput
    c. Kernels in RTOS are preemptable
        i. If high priority request comes in, they can preempt OS requests
11. OS concepts-what is mutual exclusion?
    a. When two processes try to enter a critical region one will be blocked
12. Priorities of OS programs?
    a. EL0 - general application
    b. EL1 - OS
    c. EL2 - hypervisor
    d. EL3 - security stuff
13. Describe critical section
14. https://leetcode.com/discuss/interview-question/124638/what-happens-in-the-background-from-the-time-you-press-the-Power-button-until-the-Linux-login-prompt-appears/

C/C++ Concepts

1. What is static in C?
   Static variable is one which maintains its value throughout the function invocations.
    a. A static variable inside a function keeps its value between invocations.
    b. A static global variable or a function is "seen" only in the file it's declared in.

2. What is volatile keyword?
   A volatile variable can change unexpectedly and compiler cannot make any assumptions about it
   http://www.drdobbs.com/cpp/volatile-the-multithreaded-programmers-b/184403766
3. Difference between linked list and array? when to use linked list?
   Linked list are located in heap memory and its allocated dynamically.
   Linked list relies on references where each node consists of the data and the references to the previous and next element.
   Array is a continuous block of memory stored in sequential memory.
   Arrays are index based data structure where each element associated with an index.

BASIS FOR COMPARISON	ARRAY	LINKED LIST
Basic	It is a consistent set of a fixed number of data items.	It is an ordered set comprising a variable number of data items.
Size	Specified during declaration.	No need to specify; grow and shrink during execution.
Storage Allocation	Element location is allocated during compile time.	Element position is assigned during run time.
Order of the elements	Stored consecutively my.	Stored randomly.
Accessing the element	Direct or randomly accessed, i.e., Specify the array index or subscript.	Sequentially accessed, i.e., Traverse starting from the first node in the list by the pointer.
Insertion and deletion of element	Slow relatively as shifting is required.	Easier, fast and efficient.
Searching	Binary search and linear search.	linear search.

Memory required	less.	More.
Memory Utilization	Ineffective	Efficient

-->when to use linked list

a. you need constant-time insertions/deletions from the list (such as in real-time computing where time predictability is absolutely critical)
b. you don't know how many items will be in the list. With arrays, you may need to re-declare and copy memory if the array grows too big.
c. you don't need random access to any elements
d. you want to be able to insert items in the middle of the list (such as a priority queue)

4. What are dangling pointers? where to use them?
   a. A dangling pointer points to memory that has already been freed. The storage is no longer allocated. Trying to access it might cause a Segmentation fault. A pointer pointing to a memory location that has been deleted (or freed) is called dangling pointer.
   b. A memory leak is memory which hasn't been freed, there is no way to access (or free it) now, as there are no ways to get to it anymore.
   c. Dangling pointers are not used knowingly. They must not be used at all because they result in program crash and segmentation fault. Also they eat up the memory space unnecessarily. So ONE MUST BE CAREFUL and deallocate or NULLIFY the pointer before deleting the variable pointed to by it.
5. What are structures and unions? when to use what? sizes ?
   a. Each member in struct gets own memory location in the section
   b. Size of union is the size of largest member

	STRUCTURE	UNION
Keyword	The keyword **struct** is used to define a structure	The keyword **union** is used to define a union.
Size	When a variable is associated with a structure, the compiler allocates the memory for each member. The size of structure is **greater than or equal to the sum of sizes of its members.**	when a variable is associated with a union, the compiler allocates the memory by consideringthe size of the largest memory. So, size of **union is equal to the size of largest member.**
Memory	Each member within a structure is assigned unique storage area of location.	Memory allocated is shared by individual members of union.
Value Altering	Altering the value of a member will not affect other members of the structure.	Altering the value of any of the member will alter other member values.
Accessing members	Individual member can be accessed at a time.	Only one member can be accessed at a time.
Initialization of Members	Several members of a structure can initialize at once.	Only the first member of a union can be initialized.

      i. Only one member can have a value at a time
   c. Unions are used when:

1. We create a discriminated union. That's probably what you were thinking of by "space optimization"
2. Also need an extra bit of data to know which member of the union is "alive" (has valid data in it)

6. What is free()? how does free know how much memory to deallocate?
    a. https://stackoverflow.com/questions/1957099/how-do-free-and-malloc-work-in-c

    When you malloc a block, it actually allocates a bit more memory than you asked for. This extra memory is used to store information such as the size of the allocated block, and a link to the next free/used block in a chain of block

    When you free your pointer, it uses that address to find the special information it added to the beginning (usually) of your allocated block. If you pass in a different address, it will access memory that contains garbage, and hence its behaviour is undefined (but most frequently will result in a crash)

7. What is the difference between class and object? does class or object create memory? basically learn every detail about classes and objects and just the definition.
    a. Classes are static, they just define properties of the object
    b. When a class is instantiated, it is an object and takes up memory

Class: The building block of C++ that leads to Object Oriented programming is a Class. It is a user defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object.
An Object is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated.
Examples: If channel is a class, Star Sports, BBC, and ESPN are its objects.

a class "CAR" : Its objects are Hyundai, Ford, Suzuki. It will have the same methods but different designs -> this is how you can relate objects and classes with the real world.

8. what are virtual functions? How are they used? Why are they used? When are they used? Example?
    a. To achieve runtime polymorphism
    b. They are defined in base classes - and overridden in derived classes
    c. For example, input vehicle...but don't know if car, bus, or truck

A virtual function is a member function which is declared within base class and is re-defined (Overridden) by derived class. When you refer to a derived class object using a pointer or a reference to the base class, you can call a virtual function for that object and execute the derived class's version of the function.

- Virtual functions ensure that the correct function is called for an object, regardless of the type of reference (or pointer) used for function call.
- They are mainly used to achieve Runtime polymorphism
- Functions are declared with a virtual keyword in base class.
- The resolving of function call is done at Run-time.

9. How is multiply is implemented?
    a. Set result to 0
    b. Repeat
    c. Shift 2nd multiplicand left until rightmost digit is lined up with leftmost 1 in first multiplicand
        i. Add 2nd multiplicand in that position to result
        ii. Remove that 1 from 1st multiplicand
    d. Until 1st multiplicand is zero
    e. Result is correct

10. Design an elevator system
    a. I started from thinking about how many buttons (including buttons inside the elevator, and outside for every floor) should be tracked in this system.
    b. After that, considered how to implement an ISR (interrupt service routine) and what data structure should be used (maybe a queue for the next targeted store, and a array for status of all buttons) to track them, and use these information to control the elevator.
11. How post increment works.
    a. You use the value first and then increment it.
12. One questioner asked how to modify a malloc to guarantee that it was 32-byte aligned
    a. ptr & ~0x1F will set the last 5 bits to 0, making it 32 byte aligned
13. How do breakpoints in a C program work?
    a. http://www.nynaeve.net/?p=80
14. How does a debugger work?
15. ISR & interrupt vector table
    a. ISR addresses stored in vector table
16. Float to int conversion
17. Memory map of program
    a. Output from linker script
    b. Bss is uninitialized data
    c. Data is initialized data
    d. Text is names, symbols etc
18. If we declare more number of variables than the registers available on the processor? Where they will be stored.
    a. In some memory block and they are loaded back appropriately into registers
    b. Have to remove something that's in memory at that point
19. How to handle the Generic functions , like Void pointers
    a. Function that accepts void pointers should only accept one type
20. Data Struct/Class memory padding
    a. // char 1 byte
    b. // short int 2 bytes
    c. // int 4 bytes
    d. // double 8 bytes
    e. // pointer 4 bytes in 32 bit machine, 8 in 64-bit
    f. Memory is padded to nearest multiple
    g. For optimizing memory accesses for byte addressable memory
21. Difference between Macro and Inline?
    a. Macros are used for repeating functions throughout the code, so you use #define at the top and the preprocessor typically takes care of this
    b. Inline functions tell the compiler that the function is also defined at that spot -so you save time by not jumping to another location, jumping back,
    c. Inline functions provides following advantages over macros.
        - Since they are functions so type of arguments is checked by the compiler whether they are correct or not.
        - There is no risk if called multiple times. But there is risk in macros which can be dangerous when the argument is an expression.
        - They can include multiple lines of code without trailing backslashes.
        - Inline functions have their own scope for variables and they can return a value.
        - Debugging code is easy in case of Inline functions as compared to macros.
22. How to avoid overflow/underflow
    a. Use types like long double for more precision and more bits

23. How do you send data over the network between two machines with different endianess
    a. https://www.ibm.com/developerworks/aix/library/au-endianc/index.html
    b. https://barrgroup.com/Embedded-Systems/How-To/Big-Endian-Little-Endian

Coding

1. Write a C Program to reverse the words in a sentence
    a. I used pointer arithmetic to solve the problem. First I reversed the characters in the whole sentence and then in the next step, each word was reversed.
2. Write a C program to encode bits in a 32-bit number such that, most significant 16 bits should be reversed but lower 16 bits should be untouched. Then asked to generalize this to any number of bits.
    a. Reverse all with this http://www.geeksforgeeks.org/write-an-efficient-c-program-to-reverse-bits-of-a-number/
    b. Left shift by 16 so upper 16 are reversed
    c. AND original nbr with 0x0000FFFF to zero out upper 16
    d. XOR c and d
3. Count the number of set bits in an integer
    a. Just do n &= (n-1) in a while loop and increment count, return count
    b. Subtraction of 1 from a number toggles all the bits (from right to left) till the rightmost set bit(including the rightmost set bit). So if we subtract a number by 1 and do bitwise & with itself (n & (n-1)), we unset the rightmost set bit. If we do n & (n-1) in a loop and count the no of times loop executes we get the set bit count.
    c. Beauty of the this solution is number of times it loops is equal to the number of set bits in a given integer.
    d. http://www.geeksforgeeks.org/count-set-bits-in-an-integer/
4. Bit manipulation questions - detect pattern of ones, write masks to insert pattern of ones in a 32 bit integer, swapping adjacent odd and even bits (Answer link: https://www.geeksforgeeks.org/swap-all-odd-and-even-bits/)
5. String and array manipulations - reverse string, reverse words in a string, find duplicates in an array
6. Writs
7. Find the first non-recurring character in string. i.e. input "abbcdcaea" would return "d"
    a. https://www.glassdoor.com/Interview/white-board-find-the-first-non-recurring-character-in-a-string-i-e-input-abbcdcaea-would-return-d-QTN_1942494.htm
8. Implement a queue/fifo with push/pop functionality using linked lists
9. Create a custom malloc and free function using linked lists
10. Swap the values of two pointers without a temp variable
    a. Switch with 3 x ^ y.....x=, y=, and then x-
    b. http://www.geeksforgeeks.org/swap-two-numbers-without-using-temporary-variable/
11. Write a function that determines if a given variable is a power of 2 or not
    a. ((x != 0) && !(x & (x - 1)));
        i. Works for other than 0 so need another check for 0
12. Programming problem (live) (reverse a linked list)
13. Function that takes a 2d array of a "sudoku board", and checks to make sure it is a possible board
14. Explain and describe how binary search tree works

    a. O(n) worst, O(log n) average
15. Given a list from 1 to 100, name all the different ways you can determine if there are duplicates. Which is the most efficient?
    a. Double for loop O(n^2)
    b. Sum all the elements, if it is greater than n(n+1)/2 then duplicates O(n)
    c. Add elements in a set as you go, if it already exists there are duplicates O(n)
    d. Add into a hashmap, keeping count of nbr times it shows up O(n)
    e. Sort, and compare element to element before it O(n)
16. Another way to find the sizeof and data type without using sizeof()?
    a. ((&(var)+1) - &var)
17. Write a C++ program to find a loop in a linked list.
18. Write code in C that would hash a string and deal with collision resolution by implementing a linked list. Would this code be thread safe?
19. How to change a string to integer
    a. Strtol method
20. Write a program to Delete a node , given only a pointer to the node in a Circular linked list
21. Write a own program for strstr function
    a. Returns first occurrence of str2 in str2
22. Write a program to convert a given singly Linked list to BST
    - 1) Get the Middle of the linked list and make it root.
      2) Recursively do same for left half and right half.
        a) Get the middle of left half and make it left child of the root created in step 1.
        b) Get the middle of right half and make it right child of the root created in step 1.
23. Swapping Big Endian to Little Endian and vice versa
    a. https://stackoverflow.com/questions/2182002/convert-big-endian-to-little-endian-in-c-without-using-provided-func
24. Write a program to implement memcpy()
    a. Copies memory from one location to another for x bytes
25. Delete nth end from end of linked list
26. Write a program to share a resource - https://www.thecrazyprogrammer.com/2016/09/producer-consumer-problem-c.html
27. Write a program to identify Big Endian vs Little Endian
28. Bit Manipulation questions -
    a. Reverse the bits of an integer
    b. Check the parity of a given number
    c. Swap bits in a given integer(positions are provided)
29. Write your own functions for the following -
    a. Sizeof Function
    b. Memcpy
    c. Memmove
    d. Malloc
    e. Free
30. Asked to implement a dictionary(of strings) in C(storing and searching).
31. Find the smallest and largest number in an array of large elements.

General

1. Finding the direction of a rotating disc
    a. Quadrature decoding

2. A brain teaser question where we have to find out 45 minutes with the help of two ropes. Given that one rope burns completely in 1 Hr and the rate or burning is not consistent.
    a. Burn first rope from both ends, and second rope from one end only.
    b. When First has completely burned, 30 mins will have passed and second rope will have 30 mins left on it.
    c. Now burn second rope, which has burned for 30 mins already, from both ends, this will burn a 30 minute rope at twice the speed, making it complete in 15 mins.
    d. This will be 45 minutes total.
3. What are interrupts and if you have less external interrupt pins on a processor, how to interface multiple interrupts?
4. Explain CDMA, LTE etc.
5. What is the difference between tcp and udp.
6. Call back functions
    a. Is a function that is passed as a parameter and is called after some event occurs

UART

- Universal Asynchronous Receiver/Transmitter
- Serial data
- 2 wires
    - Rx & Tx
- No clock signal to synchronize the data
    - Adds stop and start bits to identify
    - Both UARTs ports have to operate at similar baud rate
- Doesn't support multiple masters/slaves, just 2 devices

SPI

- Serial Peripheral Interface
- One master that can control multiple slaves
- MISO, MOSI, CLK, SS
- Data can be streamed continuously
  No stop/start bits

I2C

- SDA/SCL
- Synchronized to the clock signal
- Start + Address Frame + ACK + Data Frames + ACK + Stop
- Both ways transfer

CAN

- Controller Area Network
- Asynchronous Serial communication
- CSMA/CD (collision detection), but with priority levels for collision

- Normal state is around 2.5V, CANH is at 3.5V and CANL is at around 1.5V, differential signal

Bluetooth Low Energy (BLE)

- Slave connections are exclusive
    - Master can connect to multiple slaves, but slave can only connect to one master
- GATT profile
    - Services - functions
    - Characteristics - data points
- Advertising mode - sends out info to nearby peripherals
- Connected mode - one on one connection where they share data
- Range of 100m (Bluetooth 5 Coded PHYs can extend this much further)
- Point to point, mesh network (Bluetooth Mesh has a separate specification)

Advantage of mesh is that they can communicate amongst themselves z without the need for a special gateway.

4G/LTE

- CDMA uses code division multiplexing
- LTE is an IP based system (data centric)
- IP tunnel
- High speed/low latency

5G

- Goal is to simultaneously be able to connect to thousands of devices
    - Increased throughput, decreased latency
- Higher frequency for faster transmission
- Not official yet

Sorting Algorithms

Sort	Time Complexity			Space Complexity			
	Best	Worst	Average	Best	Worst	Average	
Bubble Sort	O(n^2)	O(n^2)	O(n^2)	O(1)	O(1)	O(1)	completes some pre sorting
Adaptive Bubble	O(n)	O(n^2)	O(n^2)	O(1)	O(1)	O(1)	
Selection Sort	O(n^2)	O(n^2)	O(n^2)	O(1)	O(1)	O(1)	good for small arrays
Insertion Sort	O(n^2)	O(n^2)	O(n^2)	O(1)	O(1)	O(1)	good on partially sorted data
Adaptive Insertion	O(n^2)	O(n^2)	O(n^2)	O(1)	O(1)	O(1)	
Bucket Sort	O(n)	O(n)	O(n)	O(1)	O(1)	O(1)	good for small universes

Quick Sort	O(n log n)	O(n^2)	O(n log n)	O(1)	O(log n)	O(log n)	fastest sort, good to start out and do insertion for internal loop
Merge Sort	O(n log n)	O(n log n)	O(n log n)	O(n)	O(n)	O(n)	
Heap Sort	O(n log n)	O(n log n)	O(n log n)	O(1)	O(1)	O(1)	fixdown, fixup (log n), build heap (n) heapsort(build heap and fix down from bottom)

Additional firmware interview Questions - from glassdoor

1. Swap two variables without using a temporary variable
2. How to invert all bits in a byte?
3. C function that takes two bit indices and an int, reverses the bits in the int between the two indices.
4. Name a few different type of registers.
5. What is priority inversion and methods to avoid it?
6. Disadvantage of recursive function call in C in a resource constrained environment.
7. What is an interface?
8. What is an abstract class? - C++ concepts
9. How many pins in JTAG cable? - 20
10. What techniques would you use to reduce power consumption in an embedded system?
    a. Sleep modes , refresh operation
11. What kind of data structure you will use to store data from a serial receive line?
    a. Queue - FIFO
12. Write a C program to determine endianness of a microprocessor
13. Detect loop in a linked list
14. Difference between semaphore, mutex and spinlock
15. What are some ways in which UART can have communication error?
16. Traversing Binary Tree
17. Find an element using Binary Search in array
18. Switch debouncing software logic
19. Where do global variables get stored?
20. Inter thread communication, Inter process communication
21. What is extern variable?

5 Memory Segments in C:

1. Code Segment

- The code segment, also referred as the text segment, is the area of memory which contains the frequently executed code.
- The code segment is often read-only to avoid risk of getting overridden by programming bugs like buffer-overflow, etc.
- The code segment does not contain program variables like local variable (*also called as automatic variables in C*), global variables, etc.
- Based on the C implementation, the code segment can also contain read-only string literals. For example, when you do `printf("Hello, world")` then string "Hello, world" gets created in the code/text segment. You can verify this using `size` command in Linux OS.

- Further reading

Data Segment

The data segment is divided in the below two parts and typically lies below the heap area or in some implementations above the stack, but the data segment never lies between the heap and stack area.

2. Uninitialized data segment

- This segment is also known as bss.
- This is the portion of memory which contains:
    1. Uninitialized global variables *(including pointer variables)*
    2. Uninitialized constant global variables.
    3. Uninitialized local static variables.
- Any global or static local variable which is not initialized will be stored in the uninitialized data segment
- For example: global variable `int globalVar;` or static local variable `static int localStatic;`will be stored in the uninitialized data segment.
- If you declare a global variable and initialize it as `0` or `NULL` then still it would go to uninitialized data segment or bss.
- Further reading

3. Initialized data segment

- This segment stores:
    1. Initialized global variables *(including pointer variables)*
    2. Initialized constant global variables.
    3. Initialized local static variables.
- For example: global variable `int globalVar = 1;` or static local variable `static int localStatic = 1;` will be stored in initialized data segment.
- This segment can be further classified into initialized read-only area and initialized read-write area. *Initialized constant global variables will go in the initialized read-only area while variables whose values can be modified at runtime will go in the initialized read-write area.*
- *The size of this segment is determined by the size of the values in the program's source code, and does not change at run time.*
- Further reading

4. Stack Segment

- Stack segment is used to store variables which are created inside functions (*function could be main function or user-defined function*), variable like
    1. Local variables of the function *(including pointer variables)*
    2. Arguments passed to function
    3. Return address
- Variables stored in the stack will be removed as soon as the function execution finishes.
- Further reading

5. Heap Segment

- This segment is to support dynamic memory allocation. If the programmer wants to allocate some memory dynamically then in C it is done using the `malloc`, `calloc`, or `realloc` methods.
- For example, when `int* prt = malloc(sizeof(int) * 2)` then eight bytes will be allocated in heap and memory address of that location will be returned and stored in `ptr` variable. The `ptr`variable will be on either the stack or data segment depending on the way it is declared/used.