C Programming:

1. Difference between calloc and malloc?
A: Both allocates memory dynamically. But calloc initializes the elements with 0's.

2. Static linking/libraries Vs Dynamic linking/libraries
A: Static linking is a result of linker making copy of all used library functions to one executable file. Static libraries generally in *.a format in linux and *.lib in windows.
Dynamic linking is a process of using library functions in run time while program and library is in stack. Examples of dynamic libraries are *.so libraries.

3. Difference between const & macro?
A: Macros are handles by the pre-processor. Pre-processor replaces all macro texts with their values. Where are const's are handles by the compiler for value safety. But in modern compiler there is zero time difference between these two.

4. Memory layout in C?
A: Code segment: contains executable instructions
Data segment: Contains global & static variables initialized by the program.
Stack segment: Contains programs stack.
Heap segment: Contains varibales that are allocated dynamically by the program.

5. Difference between static&extern?
A: static is a storage type assigned to either functions are values which retains it's value even control goes from where its declared. Scope of this variables are local. Initialises values to 0. extern is a storage type which have global scope. Functions or values of extern type can be accessed from other files as wel.

6.Difference between Structure and Union?
A: Both are user defines data types having same syntax. In structure each member get separate memory space so total size of structure will be addition of memory spaces of each element while total size of union will be memory space of largest member in that union. All members can be initialised when declaring a structure but only first member can be initialised when declaring a union.

7. What is Scope of Static?
A: Depends on where we put them. If we declare them inside a function/file, scope will be within that function/file but retain it's value. In C++ if we declare them then the scope will be specific to that class.

8. C compilation stages?
A: Preprocessing, compilation, assembling, linking
Preprocessing:
a) Expansion of macros
b) Removal of code comments
c) Expansion of included files.
Compiling: Conevrts whole 'C' code to assembly language
Assembling: Assembly language code is converted to machine level language in this stage.
Linking: Linking of all functions with their definitions done here because linker know each and every function definition.

9. What does linker do in exact?
A: Linker is responsible for linking all functions with their function definitions. Apart from this linker

also does some extra work. It adds some extra code which is required when pragram start and ends. For example there is code which is required for setting up the environment like passing command line arguments. This task can be verified by using **$size filename.o** and **$size filename**. Through these commands we know that how output file increases from an object file to an executable file. This is because of the code that linker adds to our proigram.

10. Write declaration of function pointer?
A: return_type (*function_name)(arguments)

11. What is callback function?
A: If a refernce of fucntion is passed as an argument to a function to call it, then it is called as callback function.
EX: Void A(){pf("hello world\n");}
void B(void(*ptr)()) { *ptr();}
main() {
void (*ptr)()=&A;
B(ptr);
}

12.Difference between static&volatile?
A: Volatile keyword instructs the compiler not to do any optimisation. Compiler will do some optimisations to variable so that value of variable will be read from the cache instead of memory. If we use volatile keyword, everytime value will be read from the memory not from the cache. Static keyword states that generally this variable have only one copy on entire file or program. Value of this variable can be retained. If we call a function generally copy of variables will be created but in static variables case, even if we call function multiple times then the value of variable will be retained instead of creating a copy.

13. What is Memory Leak? How does it happen? Is it good for the System?
A: Memory leak is a resource leak that occurs when a program incorrectly manages memory allocations. When a program doesn't deallocates the memory that is no longer needed by the program. Memory leak reduces the system performance by reducing available memory space. Memory leaks may not be serious in normal means but its huge in case of shared memory leaks between resources and when memory is limited then memory leak will have huge effect on system. Memory leaks lasts until system resets.

14. What is Static and Global Variables? Where does they stored?
A: Both are stored in data segment of the memory. Static variables can be accessed within that file/function only depends on where we declare that. Global variables can be accessed globally even these can be accessed from other files using extern keyword.

15. What is difference between local static and global static variables?
A: local static variables are declared within a function and can retain it's value but can't access out side of that function. Global static variables are declared outside of all functions and can access these variables in entire file. Both are initialised to zero. Static variables can not accesses by other files using extern keyword.

16. What is Stack? What is heap? How program execution takes place?
A: Stack is memory segment where all local variables will be stored. Stack is a LIFO structure which stores functions return address and PC values etc.., Whenever program allocates memory dynamically, that will go into heap memory segment. Heap & stack goes upwards. Heap & stack are virtually located in opposite directions of processor's memory. Whenever stack pointer met heap, memory will be exhausted.

17. What is difference b/w Array and Pointer?
A: Array is collection of similar data types. Pointers are the variables to store address of some other variables/functions.

18. Storage classes in C and limitations?
A: **auto:** default storage class for all the variables hence we don't use this keyword much.
Extern: extern tells us that variable declared somewhere else not within the file. Extern is used to access the same global variable between the files for a larger program.
Static: used to declare static variables which preserves their values even in out of scope. Hence static varibales are declared only once and memory copy will not happen. These are initialised to zero by default.
Register: used to declare register variables. Then cpu tries to allocate memory for these variables in registers of micro processor if any free register is available. If not then these values will be allocated in normal memory itself. Generally frequently used varibales will be declared with register keyword to reduce CPU running time.

19. Two static variable are declared with same name in two different files. Will compiler give any error? Explain your answer?
A: No, compiler won't throw any error. Scope of static variables will be with in that file only. Whenever we declare two static variables with same name in different files, then two variables will be created with same name in data segment. Compiler will not give any error. If those are declared as global variables then linker will give error in linking stage.

20. What are function pointers in C?
A: Similar like pointers to variables, functions also have pointers.

1. Function pointers points to the code not data unlike ptrs to variables. Pointers save address of starting line of code to be executed in that function.
2. Unlike normal pointers, we don't [de]allocate function pointers.
3. Function name returns address of that function. We can have array of function pointers.

21. Difference between strcpy and memcpy?
A: memcpy(dest, src, bytes) will copy number of specified bytes to destination irrespective of anything. strcpy(dest, src) copies src string to destination until it finds NULL terminator. Memcpy will continue to copy even if it finds NULL.

22. Whats is structure padding in C?
A: In C, in order to align data memory extra bytes are added between structure members called structure padding. #pragma pack (1) is used in C program to avoid structure padding.

23. Whats the use of inline in C?
A: inline is a keyword. We place that keyword before functions called inline functions. Whenever we use this keyword before function, all function body will be copied whenever we call that function. Thus we can save pushing of pc and all to stack before calling a function. But if we larger code in inline functions then code size will be increase because inline function body will be copied to all whenever we call those functions. Debugging of code is easy if use inline functions instead of macros.

24. What is static function?
A: Functions are global by default in C. Placing static keyword before function makes the functions static. Static functions are restricted to with in that file only.

25. What is binary search?
A: A searching algorithm which searched for particular element in sorted array by repeatedly dividing the array into two parts. Its faster when compared with linear search. The only thing is array should be sorted for this type of serach.

26. Is binary search is useful in linked lists?
A: No binary search is not useful in linked lists. Beacuse to do binary search lists should be sorted and we need to take middle element for efficiency. In case of linked lists we need to travels through each and every node like linear search. There is no efficiency like binary serach for arrays. Hence not useful for linked lists.

27. Benifits of static variables?
A: Will be initialised to zero by default. Preserves their value even after scope!

28.What is Dangling pointer?
A: Dangling Pointer is a pointer that doesn't point to a valid memory location. Dangling pointers arise when an object is deleted or deallocated, without modifying the value of the pointer, so that the pointer still points to the memory location of the deallocated memory.

29.What is difference between i++ and ++i?
1) The expression 'i++' returns the old value and then increments i. The expression ++i increments the value and returns new value.
2) Precedence of postfix ++ is higher than that of prefix ++.
3) Associativity of postfix ++ is left to right and associativity of prefix ++ is right to left.
4) In C++, ++i can be used as l-value, but i++ cannot be. In C, they both cannot be used as l-value.

30.Can a variable be both const and volatile?Yes,

31) What are the basic data types associated with C?

Int –Represent number (integer)
Float –Number with a fraction part.
Double –Double-precision floating point value
Char –Single character
Void –Special purpose type without any value.

32)**What is the behavioral difference when include header file in double quotes ("") and angular braces (<>)?**
Ans) When Header file include within double quotes (""), compiler search first in the working directory for the particular header file. If not found then in the built in the include path. But when Header file include within angular braces (<>), the compiler only search in the working directory for the particular header file.

33) Describe the modifier in C?
Ans) Modifier is a prefix to the basic data type which is used to indicate the modification for storage space allocation to a variable.
Example–In 32-bit processor storage space for int data type is 4.When we use it with modifier the storage space change as follows.
Long int ->Storage space is 8 bit
Short int ->Storage space is 2 bit
Short , Long, Signed, Unsigned, long long

34)How a negative integer is stored.?

Get the two's compliment of the same positive integer. Eg: 1011 (-5)
Step-1- One's compliment of 5:1010
Step-2-Add 1 to above, giving 1011, which is -5

35)S++ or S = S+1, which can be recommended to increment the value by 1 and why?
A: S++, as it is single machine instruction (INC) internally.

36)What is the difference between actual and formal parameters?
A: The parameters sent to the function at calling end are called as actual parameters while at the receiving of the function definition called as formal parameters.

37) Can a program be compiled without main() function?
A: Yes, it can be but cannot be executed, as the execution requires main() function definition.

38)What is reminder for 5.0 % 2?
A: Error, It is invalid that either of the operands for the modulus operator (%) is a real number.

39) What are the different ways of representing, ar array and 2D array?
A: arr[2], 2[arr], *(2+arr), *(arr+2)
2D array: arr[2][3], *(*(arr+2)+3),*(arr[2]+3)

40) Given int a[10], the operation a++ is allowed or not? Explain.
ANSWER:
Here a cant be altered that is a++ is not allowed because base address of array will be stored in read
only section(.text section).
but elements a[0],a[1] can be altered because it will be stored in stack.

41)If we just want to iterate in for loop and print it, Which one is faster and why?
a. Array
b. Link List.

Ans. Arrays. As it is stored in the contiguous memory location

42)What is a segmentation fault
A:A segmentation fault occurs when a program attempts to access a memory location that it is not allowed to access, or attempts to access a memory location in a way that is not allowed (for example, attempting to write to a read-only location, or to overwrite part of the operating system).
Reasons for segmentation fault:
a. Working on a dangling pointer and dereferencing a null pointer.
b. Writing past the allocated area on heap.
c. Operating on an array without boundary checks.
d. Freeing a memory twice.
e. Working on Returned address of a local variable
f. Running out of memory(stack or heap)

43)What is a function pointer explain its usage.
A: prototype of a function pointer: ReturnType (*PtrToFun)(arguments if any);
A function pointer is a variable which is used to hold the starting address of a functions and the same

can be used to invoke a function. It is also possible to pass address of different functions at different
times thus making the function more flexible and abstract.
a.Function pointers are mainly used in callback mechanisms.
b.Also used in Functions as Arguments to Other Functions.

43)Deallocate memory without using free() in C?
A:void *realloc(void *ptr,size_t size);
if size=0 then call to realloc is equivalent to free(ptr)

44)If we declare int i, it will occupy 2 or 4 bytes depends on the compiler. If we want to allocate 1
byte for an integer variable what can we do?
A: Can use bit fields

45)What is the difference between typedef and #define?
ANSWER:
typedef int* int_p1;
int_p1 t1, t2; // t1 and t2 are both int pointers.
#define int_p2 int*
int_p2 d1, d2; /* only the first, d1 is a pointer! as preprocessor outputs int* d1, d2 ;*/

Hence it is always advisable to use typedef to define new datatypes.

46)program to find the endianness of a processor?
Answer:
```
unsigned int i = 0xabcdef12;
unsigned char *pc = (unsigned char *) &i;
if (*pc == 0x12) {
/* little endian */
printf(“little endian \n”);
}
else {
/* big endian */
printf(“big endian \n”);
}
```

Write your own strcpy function?
Programs about static and extern variables?
WAP to remove duplicate entries from an array?
WAP to remove odd nodes from linked lists?
WAP for your malloc function?
WAP to reverse linked list?
WAP to toggle particular bits?
Write a simple Makefile Syntax for 1.c and 2.c
WAP to add node to singly linked list?
What is BST? WAP for BST?
Find and eleminate loop in linked lists?
How to insert or remove elements from BST?

Merging of two linked lists?
How malloc is implemented in C?
WAP to check if number is 2^n
WAP to swap 0xABCD to 0xCDBA
WAP to reset bit in nth position
Bitwise operations.

Protocals:

1. What happens if two i2c slaves has same slave address and master tries to communicate?
A: Theoritically there will be conflict. I2c bus sends data on device but dont know which slave may receive that data. So conflicts will araise when try to acknowledge the i2c bus same time.

2. How UART data bits are sent? How synchronization between transmission and reception happens without clock?
A: All UART pins will be high by default. When ever it wants to send some data then it pulls down that Tx pin to low(start bit), then receiver understands that it need to recieve the data. UART Tx port which is ready to send data receives parallel data from data bus. Then it adds start bit, optional parity bit and stop bit(pull up Tx line from low to high) to the data. Then sends the data to Rx end. Receiver receives the data and removes start bit, stop bit and parity bit. Then converts serial data to parallel data then keeps that data on bus. Synchronization between transmission and reception happens in UART through baudrate. Genrally baudrate will be upto 115200bps.

3.Which is better SPI or I2C? And in what aspect?
A: Depends on the use case. If speed is the matter irrespective of hardware then we can go for spi. If we wanna transfer data for longer distance then we can prefer i2c with less hardware and less speed.

4. What is clock stretching in i2c?
A: There are some cases when i2c slave is not cop-up with the i2c master's speed. Then i2c slave will not release the clock signal to high to reduce the speed of transfer. Such type of stage is called clock stretching. Its used by the slave device to reduce speed of i2c master.

5. Define each protocal? I2c, spi and UART?

I2C	SPI	UART
Inter Integrated Circuit 1. two wired protocal 2. Transmission w.r.t. Clock 3. Distance covered is more 4. support multiple masters 5. less hardware 6. startbits and stop bits are used. ACK will be there for every 8/16/32 bits to cross check if data received properly. 7.slave addresses are used to each device. 8.More than one masters can be used in the electronic circuit design. •Needs fewer i.e. only 2 wires for communication. •I2C addressing is simple which does not require any CS lines used in SPI and it is easy to add extra devices on the bus. •It uses open collector bus concept. Hence there is bus voltage flexibity on the interface bus. •Uses flow control. 9.They are suitable for communication between only two devices. •It supports fixed data rate agreed upon between devices initially before communication otherwise data will be garbled.	Serial Peripheral Interface 1. four wired protocal 2. Transmission w.r.t. Clock 3. Distance covered is more 4. Single master 5. more hardware 6.slave select line is used. 7. slave select lines are used to reach slave device. 8.It is simple protocol and hence so not require processing overheads. •Supports full duplex communication. •Due to separate use of CS lines, same kind of multiple chips can be used in the circuit design. •SPI uses push-pull and hence higher data rates and longer ranges are possible. •SPI uses less power compare to I2C 9.As number of slave increases, number of CS lines increases, this results in hardware complexity as number of pins required will increase. • To add a device in SPI requires one to add extra CS	Universal Asynchronous Receiver and Transmitter 1. two wired protocal 2. clock is not required. Baudrate is used to transmit the data 3. Distance covered is less about 50feet 4. No masters 5. lesser hardware 6. start and stop bits and optional parity bits are used. 7. No slave devices. Two devices are connected vis tx and rx pins. 8.It is simple communication and most popular which is available due to UART support in almost all the devices with 9 pin connector. It is also referred as RS232 interface. 9.They are suitable for communication between only two devices. •It supports fixed data rate agreed upon between devices initially before communication otherwise data will be garbled.

	line and changes in software for particular device addressing is concerned. •Master and slave relationship can not be changed as usually done in I2C interface. •No flow control available in SPI.	

6. Max number of slaves in I2C and Max number of Slaves in SPI
A: Depends os slave adress bits in i2c case. If we use 7/8/10 bits slave adress then we can adress 128/256/1024 devices. In SPI case we can have two slaves per SPI.

What happen when we pull up a GPIO pin externally? What's the drawback of it?
What is Quad SPI?
I2C data packet size?

OS concepts and Linux Kernel concept :

1. What is IPC? What are different types of IPC?
A: IPC stands for interprocess communication. Name itself indicating that IPC is a mechanism that allows different processes to cmmunicate with each other by sharing some data between them. IPC enables data communication by allowing the process to use segments, semaphores and other mechanisms. There are 4 types of IPC mechanisms. Pipes, FIFO, message queues, Shared memory. Shared memory is like memory will be shared between the processes and process will read data from that memory. Pipe generally invoking by pipe system calls in which data enters from one end and exits from other end. Processes can read data from the pipe only once. FIFO also works on same principle, first in first out bases. Problem with FIFOs, Queues and Pipes are whenever two processes wants to share the info, it has to through the kernel.

2. What is Virtual Memory?
A: Virtual memory is a memory management technique that abstracts actual memory resources in the system and give illusion to users of main memory. Virtual memory will be mapped to physical memory by the operating system.

3. How and why to create shared memory?
A: Shared memory is useful whenever two processes wants common memory so that those processes can view or modify similiar data. Shared memory is useful for IPC.

4. What are different syncronization menthods?
A: Mutex and Semaphores are kernel resources that provides syncronization services.
Mutex: Is a binary varivale provides locking mechanism. It will lock the critical section of the code. When being used by one thread, it won't allow another service to use same section of the code. It used for threads and used in userspace.
Semaphore: Semaphore is a signialing mechonism which provides two atomic operations called signal and wait. For example if we answer a call while listening to songs in mobile, then call process will interrupt music service that means it signals that service and stops that service temporarily. It's used for processes. Used in kernel space and its a signaling mechanism.
Spinlocks: Spimlocks keeps spinning whenever resource is busy. They go and grab the resource whenever it gets a resource.. But spinlocks consumes more number of CPU cycles. If we use spinlocks in system having one CPU then performance go bad.

5. What is Thread? Diff b/w thread and process?
A: Thread is a light weight process. A process can contain multuiple threads. The idea is to achieve parallelism by diving process into multiple threads. We can utilize all cpu's if we divide process into threads. Each thread have its own registers and stack. As soon as thread completes it's execution, it will return. Threads runs in shared memory while processes runs in seperate memory space. Context switching is faster in theads when compared with processes. Communication between threads is easy when compared with processes as threads runs in shared memory space.

6. What is deadlock?
A: Deadlock is a situation where set of processes are blocked because each process holding one resource and waiting for another resource holded by another process. Causes are:
1. Mutual Exclusion: One or more than one resources are non-sharable.
2. hold and wait: Process holding one resource and waiting for another resources.
3. No-Preemption: Process which is holding resource can not be pre empted.
4. Circular-hold: Processes are holding and waiting for another resources in circular form.

7. What is critical section? Why it is called critical section??
A: In concurrent programming, concurrent access to shared resource can lead to errorness behaviour of the code. So we protect that section where sharing of resource will happen. This protected section is called "Critical section" or "critical region". Critical section are locked by the

processes before we use them.

8.Difference between thread and process?

Thread	Process
1. Runs in shared memory	1. Runs in seperate memory.
2. Context switching is easy.	2. Context switching is difficult. Overhead on the CPU is more
3. Communication between threads is easy.	3. Communication between process are difficult.
4. Thread is a light weight program.	4. Program under execusion is process.
5. Don't have it's own segments.	5. Has it's own heap, stack,code etc segments
6. Can't live on it's own. Attached to process.	6. Have at least one thread and live on it's own.
7. If a thread dies, it's stack is reclaimed.	7. Is process dies, all threads die.

9 Whats is OS?
A: OS is a interface which will transfer user language to machine language. Is a collection of software that manages hardware and provides services to software programs. It will provide abstraction of hardware.

10. What is device tree?
A: Device tree is tree data structure where we specify device details to the kernel. Its a data structure with nodes contains physical devices in the system.

11. Why device tree is required?
A: We used to have board files before device tree to specify hardware details. With increasing number of platforms, we need to maintain lot of #ifdefs for each board which is a mess. We need to recompile whole kernel each time we modify device details in board files. Device tree provides solution to dynamically describe the hardware details to kernel running on the platform. Hence make kernel multi-platform binary. Device tree is where you collect the platform details by leaving kernel re-usable across platforms.

12. What are different types of scheduling algorithms?
A: Task scheduling is one of the imporatnat concetps of OS. There are multiple ways to schedule a task. Those are:
1. Round robin: Each task will be given certain amount of time. After that time next task will be executed irresptive of priority.
2. First come first serve: Which will come first will be served first.
3. Priority based scheduling: Higher priority task will be served first.

13. What is priority inversion?
A: Priority inversion is scenario in task scheduling where higher priority task preempted by lower priority task when ever they both are sharing common resource.

14. How many types of drivers?
A: char drivers, block drivers

15. What is differance between char driver and block driver?

Char drivers	Block drivers
1. generally not addressable. Providing data in terms of bytes. Ex: key board	1.are addressable in device specific blocks called chunks and support seeking and radom

2. abbreviated as cdev 3. accessed through a node in file system char node.	access data. 2. abbreviated as blkdev 3. Accesses through a device file node and generally mounted as file system. Ex: hard disk.

16. What is memory barrier? Why it is used?
A: Memory barriers are useful to control over order of memory access. This is necessary sometimes because optimisations performed by the compiler and hardware may be in different order sonetimes.

17. What is interrupt?
A: Interrupt is a signal to processor from hardware or software indicating that a work or event need to done immediately. Processor responds to interrupt by suspending its current activities and saving its current work by executing ISR. After executing ISR, processor resumes its works and does to normal activities.
Interrupts can be software or hardware interrupts. Software interrupts are raised by some instructions in the code. Hardware interrupts are raised by physical devices through some electrical signals.
Interrupt is an event that changes normal flow of CPU execusion.
Synchronous Interrupts or Exceptions: Produced by the CPU while processing interrupts. Ex: device errors, page defaults etc.
Ansynchronous Interrupts: Generated by other hardware devices.

18. How interrupt will be served in kernel?
A. Systems often use PIC(Programmable Interrupt Controller) to group device interrupts together before passing signal to interrupt pin of the CPU. Linux kernel maintains a table containing addresses of all ISR's. When interrupt occurs kernel will search for appropriate ISR in that table to handle the IRQ.
To register an interrupt handler we use,
int request_irq(unsigned int irq, void(*handler) (int, void struct pt_regs *),
unsgined long irq_flags, const char*dev_name, void *dev_id)

Whenever interrupt occurs processors looks if any interrupts are masked, if they are masked then nothing happens, interrupts only served if they are unmasked. Processor executes interrupt by branching particular memory address. This is called interrupt handler. Interrupt handler is something which will decide what to do when an interrupt occurs. Interrupt handler can be split into two categories. "Top halves and botton halves".
Interrupt handler first disables all interrupts. Notes the reason why that interrupt raised. Then schedule the work that need to be done. Send that work to bottom halves. Enables all interrupts. Bottom halves will do actual interrupt related work.

The interrupt handler must run quickly. Because it's preventing any other interrupt from running. In the linux kernel, interrupt processing is divided in two parts:

1. The "Top half" is the interrupt handler. It does the minimum necessary, typically communicate with the hardware and set a flag somewhere in the kernel memory.
2. The "bottom half" does any other necessary processing. For example copying data into process memory or update some data structures etc., It can take it's time and even block waiting for some other part of the syste, since it runs with interrupts enabled.

19. Types of bottom halves?
A: Soft IRQs , work queues and Tasklets

	Soft IRQs	Tasklets	Work queues
Process/interrupt context	Defered work runs in interrupt context	Defered work runs in interrupt context	Runs in process context
Concurrent execusion	Same/different soft irqs can run concurrently on different CPU's	Same tasklets cant run concurrently but different types of tasklets can run on different CPU's	Runs concurrently.
Pre emption	Can't sleep because runs in inetrrupt context	Can't sleep because runs in interrupt contaxt	Can sleep because runs in process context
Usage	Not easy to use	Can be created dynamically so Easy to use	Easy to use

20. Difference between fork() and vfork()?

A: Both are system calls which created child process that is identical to the parent process.

Basis of comparision	fork()	vfork()
Basic	Child process and parent process have different address space.	Child process and parent space have same address space.
Execusion	Both child and parent processes will be executed simultaneously.	After executing child process parent process will be executed.
Modification	If the child process alters any page in the memory it is unvisible to parent process. Because both processes use different address space	If the child process alters any page then it will be visible to parent process because both process shares same address space.
Copy on write	Uses copy on write as an alternative where parent and child uses same pages until one of them modifies the shared page.	Does not support.

21. How system call work?

A: System call is a medium or path through which user space programs can request kernel services. Some of the system calls are fork(), getpid, getppid, wait, exit, open(), read(), write() etc.,

From a programmer point of view, calling a system call is equivalent to calling any normal C function.

Inside the Linux kernel, each system call has a number associated with it.

The kernel expects all the information like system call number, the arguments to the system call etc to be present in CPU registers.

A system call first invokes a C library wrapper function which does all this work of placing the information required to be placed in CPU registers. The system call number is placed in the register %eax.
After doing the above work, the same wrapper function invokes a trap instruction 'int 0x80'. This instruction switches the processor mode from user mode to the kernel mode and executes the code kept at location 0x80. Note : Starting with Linux kernel 2.6 and glibc 2.3.2 a faster mechanism of using the 'sysenter' instruction is followed to switch into the kernel mode.
After the above step, now since the processor is in kernel mode, the kernel calls a general trap handler function system_call() to handle this trap.
This handler function : Copies all the relevant information kept in the CPU registers onto the kernel stack, Validates the system call number, Uses the system call number to find the corresponding system call service routine from a table(sys_call_table) of service routines maintained by kernel and Calls the system call service routine with the arguments passed from the user space.
The system call service routine that is finally called by the trap handler first checks the validity of the arguments sent to it and then it does the actual work requested by the application.
Once the system call service routine is done with its processing, the control is returned back to the trap handler function which returns the flow back to the wrapper function in the user-mode.
Based on the return value from the trap handler, the wrapper function sets the 'errno' variable and returns -1 (in most of the cases) to indicate an error.
The value of the 'errno' variable is assigned by negating the return value of the trap handler (returned to wrapper function).
The application can use the strerror() API with 'errno' to get the error information in form of human readable string.

22. Difference between kmalloc() and malloc()?
A: malloc can be called from userspace as well as from kernel space. It really does not allocate memory physically. Physicall memory will be mapped later. Where as kmalloc() will allocate memory physically. Will be called from kernel only.

23.what is deadlock? causes? ways to avoid

Ans :
- Is it a state where two or more operations are waiting for each other, say a computing action 'A' is waiting for action 'B' to complete, while action 'B' can only execute when 'A' is completed. Such a situation would be called a deadlock.

In order for deadlock to occur, four conditions must be true.
- Mutual exclusion - At least one resource must be non-shareable.[1] Only one process can use the resource at any given instant of time
- Hold and Wait - processes currently holding resources can request new resources
- No preemption - Once a process holds a resource, it cannot be taken away by another process or the kernel.
- Circular wait - Each process is waiting to obtain a resource which is held by another process.

- Deadlock can be avoided if certain information about processes are available to the operating system before allocation of resources, such as which resources a process will consume in its lifetime

24. Interprocess Communication

🎬 Interprocess Communication (IPC) is a mechanism by which two or more processes communicate with each other to exchange data by synchronizing their activities.
🎬 PC uses read() and write() and some Input/Output (I/O) system functions communication.
🎬 IPC provides a programming interface that allows you to create and manage processes, which can run concurrently in an Operating System (OS).

25. IPC Types
In Unix, IPC enables communication between processes either on the same computer or on different computers. The
different types of IPC available are:
🎬Local IPC: Allows one or more processes to communicate with each other on a single system as follows:

o Pipe: Passes information from one process to another process. There are two types of pipes, named pipes and unnamed pipes.An unnamed pipe is a one-way communication IPC whereas a named pipe is a two-way communication IPC.
o Signals: Are software-generated interrupts sent to a process when an event occurs.
o Message queuing: Allows processes to exchange data by using a message queue.
o Semaphores: Synchronizes the processes activities concurrently competing for the same system resource.
o Shared memory: Allows memory segments to be shared among processes communicating with each other to exchange data between them. This method of IPC is used for faster communication between processes than the reading and writing of data by other operating system services.
o Sockets: Are end-points for communication between processes. They allow communication between a client program and a server program.

26.Priority Inversion (what, where and why)
Ans:
priority inversion is a problematic scenario in scheduling in which a high priority task is indirectly preempted by a medium priority task effectively "inverting" the relative priorities of the two tasks.
Consider a task L, with low priority, that requires a resource R. Now, consider another task H, with high priority. This task also requires resource R. If H starts after L has acquired resource R, then H has to wait to run until L relinquishes resource R.
Everything works as expected up to this point, but problems arise when a new task M (which does not use R) starts with medium priority during this time. Since R is still in use (by L), H cannot run. Since M is the highest priority unblocked task, it will be scheduled before L. Since L has been preempted by M, L cannot relinquish R. So M will run until it is finished, then L will run - at least up to a point where it can relinquish R - and then H will run. Thus, in the scenario above, a task with medium priority ran before a task with high priority, effectively giving us a priority inversion.
Priority inheritance:
priority inheritance is a method for eliminating priority inversion problems. Using this programming method, a process scheduling algorithm will increase the priority of a process to the maximum priority of any process waiting for any resource on which the process has a resource lock.
The basic idea of the priority inheritance protocol is that when a job blocks one or more high priority jobs, it ignores its original priority assignment and executes its critical section at the highest priority level of all the jobs it blocks. After executing its critical section, the job returns to its original priority level. Consider three jobs:

Job Name	**Priority**
H	**High**
M	**Medium**

L Low
Suppose H is blocked by L for some shared resource. The priority inheritance protocol requires that L executes its critical section at the (high) priority of H. As a result, M will be unable to preempt L and will be blocked. That is, the higher priority job M must wait for the critical section of the lower priority job L to be executed, because L now inherits the priority of H. When L exits its critical section, it regains its original (low) priority and awakens H (which was blocked by L). H, having high priority, immediately preempts L and runs to completion. This enables M and L to resume in succession and run to completion.

27. Fragmentation.(The process or state of breaking or being broken into small or separate parts.)

- Fragmentation is a phenomenon in which storage space is used inefficiently .
- When a program is started, the free memory areas are long and contiguous. Over time and with use, the long contiguous regions become fragmented into smaller and smaller contiguous areas. Eventually, it may become impossible for the program to request large chunks of memory.

Types of Fragmentation :
1. Internal fragmentation

- Due to the rules governing memory allocation, more computer memory is sometimes allocated than is needed. For example, memory can only be provided to programs in chunks divisible by 4, 8 or 16, and as a result if a program requests perhaps 23 bytes, it will actually get a chunk of 24. When this happens, the excess memory goes to waste. In this scenario, the unusable memory is contained within an allocated region, and is thus termed internal fragmentation

2. External fragmentation

- External fragmentation arises when free memory is separated into small blocks and is interspersed by allocated memory.
- The result is that, although free storage is available, it is effectively unusable because it is divided into pieces that are too small individually to satisfy the demands of the application.

3. Data fragmentation

- Data fragmentation occurs when a collection of data in memory is broken up into many pieces that are not close together. It is typically the result of attempting to insert a large object into storage that has already suffered external fragmentation.
- system puts the new data in new non-contiguous data blocks to fit into the available holes. The new data blocks are necessarily scattered, slowing access due to seek time and rotational latency of the read/write head.

There are a variety of algorithms for selecting which of those potential holes :
- The "best fit" algorithm chooses the smallest hole that is big enough.
- The "worst fit" algorithm chooses the largest hole.
- The "first-fit algorithm" chooses the first hole that is big enough.
- The "next fit" algorithm keeps track of where each file was written.

The "next fit" algorithm is faster than "first fit", which is in turn faster than "best fit", which is the same speed as "worst fit"

28. Difference between interrupt & signal?
Interrupts can be viewed as a mean of communication between the CPU and the OS kernel.
Signals can be viewed as a mean of communication between the OS kernel and OS processes.

Interrupts may be initiated by the CPU (exceptions - e.g.: divide by zero, page fault), devices (hardware interrupts - e.g: input available), or by a CPU instruction (traps - e.g: syscalls, breakpoints). They are eventually managed by the CPU, which "interrupts" the current task, and invokes an OS-kernel provided ISR/interrupt handler.
Signals may be initiated by the OS kernel (e.g: SIGFPE, SIGSEGV, SIGIO), or by a process(kill()). They are eventually managed by the OS kernel, which delivers them to the target thread/process, invoking either a generic action (ignore, terminate, terminate and dump core) or a process-provided signal handler.

29. What is cache memory?
A: Cache memory, also called CPU memory, is high-speed static random access memory (SRAM) that a computer microprocessor can access more quickly than it can access regular random access memory (RAM). This memory is typically integrated directly into the CPU chip or placed on a separate chip that has a separate bus interconnect with the CPU. The purpose of cache memory is to store program instructions and data that are used repeatedly in the operation of programs or information that the CPU is likely to need next. The computer processor can access this information quickly from the cache rather than having to get it from computer's main memory. Fast access to these instructions increases the overall speed of the program. There are L1, L2, L3 caches available. If CPU doesn;t find required data in cache there will be cache miss so it will go and get the content from main memory.

30. If two process trying to read & write the data in cache then how will you save the data from corruption?

22. Memory management in linux?

Pthread creation and how pthread_wait and pthread_join works?
How to create process and thread?
What data is shared between two processes?

Round 1

7. If two process trying to read & write the data in cache then how will you save the data from corruption?

Round 2

1. How to debug memory leak in code?
2. How to find a stack overflow in memory?
3. What are the registers in microprocessor? Explain in details
4. What happen in assembly language programming when we call a function?
8. How to design a random function to select a song from playlist which should not repeat again

1.How to design a random function to select a song from playlist which should not repeat again ?

2. Describe about Paging Concept ?
6.WAP for fibonacci,factorial using recursion ?

1. Describe Integer overflow ?
5.If two process trying to read & write the data in cache then how will you save the data from corruption?
6. 4 threads (1-2-3-4 ,3 will kill signal 1 is there any issue) ?

7. Predict the O/p

```
int main(){
static int i=4;
if(--i){
main();
}
printf("the value is %d",i);
}
```

8. float to string ex: 531.12 in WAP that takes float as an argument that has to be transferred into string and print that string ?
7)In matrix arr[10][20], what is the address of arr[3][5] if starting address is 1000??
8)implementation of dynamic 2D array.
9)What is the output and why, with memory layout.?

```
void fun(){
Static int i=10;
i++;
printf("%d",i);
}
main(){
fun();
fun();
}
```

Kernel initialization process ? Where MBR (Master boot record) will be initialized ?
Device driver memory issues,
IO Remap call without io_free in driver
copy from user and copy to user when working with user and kernel buffers
Request IRQ api, ISR function flow
kernel memory allocation api's
Structure of platform drivers
I2C drivers

About TLB's , page_fault , caches ?

How virtual file system will work in the linux system? (/sys , /proc/ /dev)

Bus devices and different classes in sys

RISC vs CISC

Code density of RISC and CISC , which implementaion have more code density ?

mknod usage ? cdev infrastructure ?

Function of module_init ?

Explain about basic driver registation (platform and basic char driver)

Do we have any idea abt device tree ? What is the advantage of using it ?

when probe function will be called ?

Interrupt mechanism in linux

TOP and Bottom halves ? Explain ? (tasklets , softirqs , schedulers) , concurrency mechanism in these techniques ?

Where vector table will be stored (in downward direction or upward) - 0x0000 0000

Can we implement softirqs in the modules (can we compile and load it dynamically) . Because , while compilation one header file will be created (softirs.h) . If we compile it as module that file won't be update
micro kernel vs monolithic kernel ? (RTOS - micro kernel , Android - Hydrid kernel , linux - monolithic kernel)
TRAP , kill 3
Memory pools in kernel
I2c multi master case
NAND vs NOR fault tolereances
Uboot flow with functions

Device Drivers:

1. What Is Mknod And It's Usage ?
Answer :
mknod is a command which used create the device file (or) node in Linux file system.
In unix or linux we will represent everything as a file .
syntax:mknod Name { b | c } Major Minor
Name : name of the device file
{ b | c } : type of device (ex; char or block device)
Major : Major number of the device file
Minor : Minor number of the device file
ex :$ mknod /dev/rama c 12 5
MKDEV(int major, int minor);

2. In How Many Ways We Can Allocate Device Number ?
Answer :
In 2 ways we can allocate device numbers
statically
dynamically

3. How Can We Allocate Device Number Statically ?
Answer :
register_chrdev_region() function will statically allocate device numbers. which is declared in <linux/fs.h>
int register_chrdev_region(dev_t first, unsigned int count, char *name);
Return values : In case of success "0" will return , In case of failure "-1 or negative value " will return
Here
first is the beginning device number of the range you would like to allocate. The minor number portion of first is often 0.
count is the total number of contiguous device numbers you are requesting.
name is the name of the device that should be associated with this number range. it will appear in /proc/devices and sysfs.

4. How Can We Allocate Device Number Dynamically ?
Answer :
alloc_chrdev_region()will dynamically allocate device numbers.
int alloc_chrdev_region(dev_t *dev, unsigned int firstminor,
unsigned int count, char *name);
Here
dev is an output-only parameter that will, on successful completion, hold the first number in your allocated range.
firstminor should be the requested first minor number to use; it is usually 0
count is the total number of contiguous device numbers you are requesting.
name is the name of the device that should be associated with this number range. it will appear in

/proc/devices and sysfs.

5. How Can We Free Device Numbers ?
Answer :
void unregister_chrdev_region(dev_t first, unsigned int count);

6. What Is Major Number And It's Usage ?
Answer :
It's an integer number mainly used to provide the association between the device driver and device file . this number is used by kernel .
(or)
The major number tells you which driver handles which device file.

7. Can We Have Same Major Number For More Than One Device File ?
Answer :
yes . we can have .

8. What Is Minor Number And It's Usage ?
Answer :
The minor number is used only by the driver itself to differentiate which device it's operating on, just in case the driver handles more than one device.
(or)
one driver can control more than one device .minor will be used to distinguish the one device from other devices .

9. What Is Range Of Major And Minor Numbers?
Answer : 0-255

10. What Is Use Of Dev_t Type ?
Answer :
This is used to hold device numbers—both the major and minor parts.

11. How To Retrieve Major And Minor Number From Dev_t Type ?
Answer :
To obtain the major or minor number of a dev_t, use:
MAJOR(dev_t dev);// to obtain major number
MINOR(dev_t dev);// to obtain minor number
int major=MAJOR(dev_t dev);
int minor =MINOR(dev_t dev);

12. How Can I Use My Own Major And Minor Number For A Device File ?
Answer :
if you have the major and minor numbers and need to turn them into a dev_t, use:
register_chrdev_region works well if you know ahead of time exactly which device numbers you want. Often, however, you will not know which major numbers your device will use; there is a constant effort within the Linux kernel development community to move over to the use of dynamically-allocated device numbers.

13. How To See Statically Assigned Major Numbers ?
Answer :
Some major device numbers are statically assigned to the most common devices. A list of those devices can be found in Documentation/devices.txt within the kernel source tree.

14. What Is The Disadvantage Of Dynamic Device Number Assignment ?

Answer :

The disadvantage of dynamic assignment is that you can't create the device nodes in advance, because the major number assigned to your module will vary.

15. Where Can We Write Allocation And Freeing Of Device Number's Code ?

Answer :

allocation :init function of a module

freeing :cleanup function of a module

Round 1

5. What is cache memory?
6. Difference between semaphore & mutex with example
7. If two process trying to read & write the data in cache then how will you save the data from corruption?
8. Android Architecture in detail
9. Explain Memory layout in C with example
10. Convert string into float
11. Output related questions in static & char pointer

Round 2

1. How to debug memory leak in code?
2. How to find a stack overflow in memory?
3. What are the registers in microprocessor? Explain in details
4. What happen in assembly language programming when we call a function?
5. Explain storage class in C
6. Stack and Heap details in memory layout in C
7. What is volatile and its uses
8. How to design a random function to select a song from playlist which should not repeat again
9. What are components in Openmax?
10. Component status in Openmax with block diagram
11. Openmax API’s used to load & communication in components

Prateek Srivastava

Round 1

1.How to design a random function to select a song from playlist which should not repeat again ?
2. Describe about Paging Concept ?
3 Draw memory layout and description ?
4.Difference between malloc,calloc and realloc ?
5. Describe semaphore and mutex and its use cases ?
6.WAP for fibonacci,factorial using recursion ?
7.Describe sorting and searching ?
8.Describe use cases of searching (Binary search & linear search) ?
9.Describe use cases of sorting (Bubble & Selection) ?
10. Describe goto and its use cases ?
11. Describe switch case program ?
12. Describe Video Encoder Block Diagram ?
13. Why we are using Quantization?
14. Project Related Questions(From Previous Company)

Round 2

1. Describe Integer overflow ?
2. Describe difference b/w Interrupt and signal ?
3.Describe Cache memory ?
4. Describe Difference b/w semaphore and mutex with example ?
5.If two process trying to read & write the data in cache then how will you save the data from corruption?
6. 4 threads (1-2-3-4 ,3 will kill signal 1 is there any issue) ?

7. Predict the O/p

```
int main(){
static int i=4;
if(--i){
main();
}
printf("the value is %d",i);
```

}

8. float to string ex: 531.12 in WAP that takes float as an argument that has to be transferred into string and print that string ?

Parth shah:-

Round 1:-

1)What is image processing.
2)JPEG image encoding.
3) Different type of Storage class.
4)static storage class implementation. With memory layout.?
5)what are the operations perform on Pointers with pointers(addition,subtraction,comparison)?
6)How matrix store in memory location.?
7)In matrix arr[10][20], what is the address of arr[3][5] if starting address is 1000??
8)implementation of dynamic 2D array.
9)What is the output and why, with memory layout.?

```
void fun(){
Static int i=10;
i++;
printf("%d",i);
}
main(){
fun();
fun();
}
```