Difference between AVR, ARM, 8051 and PIC Microcontrollers

The differences between the microcontrollers are mainly include what is a microcontroller, difference between AVR, ARM, 8051 and PIC microcontrollers and its applications.

What is a Microcontroller?

A micro-controller can be comparable to a little stand alone computer; it is an extremely powerful device, which is able of executing a series of pre-programmed tasks and interacting with extra hardware devices. Being packed in a tiny integrated circuit (IC) whose size and weight is regularly negligible, it is becoming the perfect controller for as robots or any machines required some type of intelligent automation. A single microcontroller can be enough to manage a small mobile robot, an automatic washer machine or a security system. Several microcontrollers contains a memory to store the program to be executed, and a lot of input/output lines that can be a used to act jointly with other devices, like reading the state of a sensor or controlling a motor.

8051 Microcontroller

8051 microcontroller is an 8-bit family of microcontroller is developed by the Intel in the year 1981. This is one of the popular families of microcontroller are being used all across the world. This microcontroller was moreover referred as “system on a chip” since it has 128 bytes of RAM, 4Kbytes of a ROM, 2 Timers, 1 Serial port, and 4 ports on a single chip. The CPU can also work for 8bits of data at a time since 8051 is an 8-bit processor. In case the data is bigger than 8 bits, then it has to be broken into parts so that the CPU can process easily. Most manufacturers contain put 4Kbytes of ROM even though the number of ROM can be exceeded up to 64 K bytes.

The 8051 has been in utilized in a wide number of devices, mostly because it is easy to integrate into a project or make a device approximately.

The following are the major areas of focus:

Energy Management: Efficient metering systems facilitate in controlling energy usage in homes and manufacturing applications. These metering systems are prepared capable by incorporating microcontrollers.

Touch screens: A high number of microcontroller providers incorporate touch-sensing capabilities in their designs. Portableelectronics such as cell phones, media players and gaming devices are examples of microcontroller-based touch screens.

Automobiles: The 8051 finds wide taking in providing automobile solutions. They are broadly used in hybrid vehicles to handle engine variants. Furthermore, functions such as cruise control and anti-brake system have been prepared more capable with the use of microcontrollers.

Medical Devices: Moveable medical devices such as blood pressure and glucose monitors use microcontrollers will to show
data, thus provided that higher reliability in providing medical results.

PIC Microcontroller Peripheral Interface Controller (PIC) is microcontroller developed by a Microchip, PIC microcontroller is fast and simple to implement program when we contrast other microcontrollers like 8051. The ease of programming and simple to interfacing with other peripherals PIC become successful microcontroller.

PIC Microcontroller We know that microcontroller is an integrated chip which is consists of RAM, ROM, CPU, TIMER and COUNTERS. The PIC is a microcontroller which as well consists of RAM, ROM, CPU, timer, counter, ADC (analog to digital converters), DAC (digital to analog converter). PIC Microcontroller also support the protocols like CAN, SPI, UART for an interfacing with additional peripherals. PIC mostly used to modify Harvard architecture and also supports RISC (Reduced Instruction Set Computer) by the above requirement RISC and Harvard we can simply that PIC is faster than the 8051 based controllers which is prepared up of Von-Newman architecture.

AVR Microcontroller

AVR microcontroller was developed in the year of 1996 by Atmel Corporation. The structural design of AVR was developed by the Alf-Egil Bogen and Vegard Wollan.

AVR derives its name from its developers and stands for Alf-Egil Bogen Vegard Wollan RISC microcontroller, also known as Advanced Virtual RISC. The AT90S8515 was the initial microcontroller which was based on the AVR architecture, though the first microcontroller to hit the commercial market was AT90S1200 in the year 1997.
AVR Microocntroller
AVR Microcontrollers are Available in three Categories
TinyAVR:- Less memory, small size, appropriate just for simpler applications
MegaAVR:- These are the mainly popular ones having a good quantity of memory (up to 256 KB), higher number of inbuilt peripherals and appropriate for modest to complex applications.
XmegaAVR:- Used in commercial for complex applications, which need large program memory and high speed.

ARM Processor

An ARM processor is also one of a family of CPUs based on the RISC (reduced instruction set computer) architecture developed by Advanced RISC Machines (ARM).ARM Microcontroller An ARM makes at 32-bit and 64-bit RISC multi-core processors. RISC processors are designed to perform a smaller number of types of computer instructions so that they can operate at a higher speed, performing extra millions of instructions per second (MIPS). By stripping out unnecessary instructions and optimizing pathways, RISC processors give outstanding performance at a part of the power demand of CISC (complex instruction set computing) procedure.
ARM processors are widely used in customer electronic devices such as smart phones, tablets, multimedia players and other mobile devices, such as wearables. Because of their reduced to instruction set, they need fewer transistors, which enable a smaller die size of the integrated circuitry (IC). The ARM processors, smaller size reduced difficulty and lower power expenditure makes them suitable for increasingly miniaturized devices.

Tell me about yourself

Good morning/afternoon/evening" sir/mam.
First of all thank you for giving me this opportunity to introduce myself.

My name is Ajeet Kumar.

As far as my education qualification is concerned, I have done B.Tech with electrical and electronics eng. from JNTU university in Anatapur, Mother Therease College of engineering, palamaner, in the year of 2015.

I have completed Diploma from YC JAMES YEN RURAL polytechnic college Kuppam in 2012.

I have completed my schooling from Z.P.high school Arikela

As par as my family is concerned, I belong to a middle class family. My father is a Former .My Mother is home maker , my brother is a civil engineer, And sister is doing MBA.

I am good in programming languages C, C++, and much interested in linux.

My strength is self confidence, positive attitude, hard work.

My weakness is : I can easily believe every one

My hobbies are: Listening music.

Volatile:-

volatile keyword is used to prevent the compiler to optimize a variable ,which can be changed unexpectedly beyound compiler comprehensation

INTERRUPT:-

It is an event in a CPU that causes the CPU to stop exicuting the current progrmm code begin the exicution of Special pice of code called as Interrupt handler.

Interrupt latency:-

iterrupt latency is the "<u>time requied for an ISR responds to an interrupt."</u>

<u>Following steps could be followed to reduce the latency</u>

- ISRs being simple and short.
- Interrupts being serviced immediately
- Avoiding those instructions that increase the latency period.
- Also by prioritizing interrupts over threads.
- Avoiding use of inappropriate APIs.

what is spin lock?

If a Resource is locked , A thread that wants to access that resource may Repetively check whether the "Resource" is avaliable . During that time the thread may loop and check the resource without doing any usfull work such a LOCK is termed as "SPIN LOCK".

detailed concept of semaphores and difference between binary semaphore and mutex?

- MUTual EXclusion and synchronization can be done by binary semaphore.

    - Mutex is used only for MUTual Exclusion.

- Semaphore value can be changed by other THREAD also.

    - A mutex can be relesed by the same thread which acquired it .

- Adv of semphore is that they can be used to synchronize two unrelated processes trying to access the same resource

    - From an ISR a Mutex can NOT be used , beacuse of another proccess goes to sleep..

"Semaphore are the Synchronization tool to overcome critcal section problem".
"Mutex is also a tool that is used to provide deadlock free mutual exclusion".

Semaphore: Use a semaphore when you (thread) want to sleep till some other thread tells you to wake up. Semaphore 'down' happens in one thread (producer) and semaphore 'up' (for same semaphore) happens in another thread (consumer)

e.g.: In producer-consumer problem, producer wants to sleep till at least one buffer slot is empty - only the consumer thread can tell when a buffer slot is empty.

Mutex: Use a mutex when you (thread) want to execute code that should not be executed by any other thread at the same time. Mutex 'down' happens in one thread and mutex 'up' *must* happen in the same thread later on.

e.g.: If you are deleting a node from a global linked list, you do not want another thread to muck around with pointers while you are deleting the node. When you acquire a mutex and are busy deleting a node, if another thread tries to acquire the same mutex, it will be put to sleep till you release the mutex.

Spinlock: Use a spinlock when you really want to use a mutex but your thread is not allowed to sleep.

e.g.: An interrupt handler within OS kernel must never sleep. If it does the system will freeze / crash. If you need to insert a node to globally shared linked list from the interrupt handler, acquire a spinlock - insert node - release spinlock.

Atomic Operations:-

Atomic operations provide instructions that execute atomicallywithout interruption.

The declarations needed to use the atomic integer operations are in **<asm/atomic.h> .**

Defining an atomic_t is done in the usual manner. Optionally, you can set it to an initial value:

```
atomic_t v;             /* define v */
atomic_t u = ATOMIC_INIT(0);         /* define u and initialize it to zero */
```

Operations are all simple:

```
atomic_set(&v, 4); /* v = 4 (atomically) */
atomic_add(2, &v);         /* v = v + 2 = 6 (atomically) */
atomic_inc(&v);    /* v = v + 1 = 7 (atomically) */
```

If you ever need to convert an atomic_t to an int , use atomic_read() :

```
printk("%d\n", atomic_read(&v)); /* will print "7" */
```

A common use of the atomic integer operations is to implement counters. Protecting a sole counter with a complex locking scheme is silly, so instead developers use **atomic_inc()** and **atomic_dec() ,** which are much lighter in weight.

Another use of the atomic integer operators is atomically performing an operation and testing the result. A common example is the atomic decrement and test:

int **atomic_dec_and_test(atomic_t *v)**

This function decrements by one the given atomic value. If the result is zero, it returns true; otherwise, it returns false.

Basics

The Four Stages of Compiling a C Program

Compiling a C program is a multi-stage process. At an overview level, the process can be split into four separate stages: Preprocessing, Translator , assembly, and linking.

preprocessing

->It includes header files,it removies the comments ,it replace with macros..

compile up to preprocessor stage

```
cc -E hello_world.c
                    -E : stop after preprocessor stage .
```

Translating

It will chaeck the syntactical errors ,it convert c program into assemble language.

compile up to Translator stage

```
cc -S hello_world.c                    :- compli translator stage
```

Assembly

During the assembly stage, an assembler is used to translate the assembly instructions to machine code, or *object code*. The output consists of actual instructions to be run by the target processor.

To save the result of the assembly stage, pass the -c option to cc:

```
cc -c hello_world.c
```

Running the above command will create a file named hello_world.o, containing the object code of the program. The contents of this file is in a binary format and can be inspected using hexdump or od by running either one of the following commands:

```
hexdump hello_world.o
od -c hello_world.o
```

Linking

It is linking with library , it will add some OS related information.

```
cc -o hello_world hello_world.c
```

A program in C language involves into different processes. Below diagram will help you to understand all the processes that a C program comes across.

There are 4 regions of memory which are created by a compiled C program. They are,

1.**First region** – This is the memory region which holds the executable code of the program.

2.**2nd region** – In this memory region, global variables are stored.
3.**3rd region** – stack
4.**4th region** – heap

Explain the types of Errors in C Programming.

Run Time Errors:

These errors are usually caught by the compilers and occur due to illegal operations performed within a program such as Dividing an Integer by Zero, Unavailability of Memory Space and others. These errors terminate the program abruptly.

Compile Time Errors:

Compilation Errors are those that occurs at the Compilation Time of a program. These errors are further divided into:

Semantic Errors

These errors occur due to undefined operations such as illegal assignment as this x+y=z.

Syntax Errors

These errors occur if we don't follow the guidelines and rules prescribed by that particular language.

Logical Errors

Logical errors are most difficult to debug as these are not usually caught by the Compiler. These generally occur due to Algorithm and Logic issues within the program.

storage classes

A storage class defines the scope (visibility) and life time of variables and/or functions within a C Program.
There are following storage classes which can be used in a C Program

- auto
- register
- static
- extern

auto - Storage Class

Auto is the default storage class for all local variables.

```
{
   int Count;
   auto int Month;
}
```

The example above defines two variables with the same storage class. auto can only be used within functions, i.e. local variables.

When is it required to use an Auto Data type for a variable?

This is probably the most common data type used for any variable. The prefix auto is not mandatory. If +you don’t have any special need of a variable, automatic storage class is the one that you should opt for.

register - Storage Class

Register is used to define local variables that should be stored in a register instead of RAM. This means that the variable has a maximum size equal to the register size (usually one word) and cant have the unary '&' operator applied to it (as it does not have a memory location).

```
{
   register int  Miles;
}
```

Register should only be used for variables that require quick access - such as counters.

When is it required to use a Register Data type for a variable?

Since there are few CPU registers available in the memory, register storage class should be used only when a variable is being used very often

static - Storage Class

Static is the default storage class for global variables. The two variables below (count and road) both have a static storage class.

```
static int Count;
int Road;

{
   printf("%d\n", Road);
    }
```

static variables can be 'seen' within all functions in this source file. At link time, the static variables defined here will not be seen by the object modules that are brought in.

When is it required to use a Static Data type for a variable?

A Static storage class should be used only when you want the value of a variable to be persistent between different functions of the same program

static can also be defined within a function. If this is done the variable is initalised at run time but is not reinitalized when the function is called. This inside a function static variable retains its value during vairous calls.

```
void func(void);

static count=10; /* Global variable - static is the default */
```

```
main()
{
  while (count--)
  {
     func();
  }

}

void func( void )
{
  static i = 5;
  i++;
  printf("i is %d and count is %d\n", i, count);
}
```

This will produce following result

```
i is 6 and count is 9
i is 7 and count is 8
i is 8 and count is 7
i is 9 and count is 6
i is 10 and count is 5
i is 11 and count is 4
i is 12 and count is 3
i is 13 and count is 2
i is 14 and count is 1
i is 15 and count is 0
```

NOTE : Here keyword *void* means function does not return anything and it does not take any parameter. You can memoriese void as nothing. static variables are initialized to 0 automatically.

extern - Storage Class

Extern is used to give a reference of a global variable that is visible to ALL the program files. When you use 'extern' the variable cannot be initalized as all it does is point the variable name at a storage location that has been previously defined.

When you have multiple files and you define a global variable or function which will be used in other files also, then *extern* will be used in another file to give reference of defined variable or function. Just for understanding *extern* is used to decalre a global variable or function in another files.

File 1: main.c

```
int count=5;

main()
{
  write_extern();
}
```

File 2: write.c

```
void write_extern(void);

extern int count;

void write_extern(void)
{
  printf("count is %i\n", count);
}
```

Here *extern* keyword is being used to declare count in another file.

Now compile these two files as follows

```
gcc main.c write.c -o write
```

This fill produce *write* program which can be executed to produce result.

Count in 'main.c' will have a value of 5. If main.c changes the value of count - write.c will see the new value

Definition vs Declaration : Before proceeding, let us understand the difference between *defintion* and *declaration* of a variable or function. Definition means where a variable or function is defined in realityand actual memory is allocated for variable or function. Declaration means just giving a reference of a variable and function. Through declaration we assure to the complier that this variable or function has been defined somewhere else in the program and will be provided at the time of linking. In the above examples *char *func(void)* has been put at the top which is a declaration of this function where as this function has been defined below to *main()* function.

There is one more very important use for 'static'. Consider this bit of code.

```
char *func(void);

main()
{
   char *Text1;
   Text1 = func();
}
```

```
char *func(void)
{
  char Text2[10]="martin";
  return(Text2);
}
```

Now, 'func' returns a pointer to the memory location where 'text2' starts BUT text2 has a storage class of 'auto' and will disappear when we exit the function and could be overwritten but something else. The answer is to specify

```
static char Text[10]="martin";
```

The storage assigned to 'text2' will remain reserved for the duration if the program.

Compare Automatic, Register, Static and External Variables in C

Let us understand the difference between various storage classes in C programming language. Below is the table that will show difference between Automatic, Register, Static and External Variables storage Classes in C.

	Feature	**Automatic Variable**	**Register Variable**	**Static Variable**	**External Variable**
1	**Keyword Used**	auto	register	static	extern
2	**Storage**	Memory	CPU registers	Memory	Memory
3	**Default initial value**	Garbage Value	Garbage Value	Zero Value	Zero Value
4	**Scope**	Local to the block in which the variable is defined	Local to the block in which the variable is defined	Local to the block in which the variable is defined	Global
5	**Life**	Till the control remains within the block in which the variable is defined	Till the control remains within the block in which the variable is defined	Value of the variable persists between different function calls	As long as the program’s execution doesn’t come to an end
6	**Use**	General purpose use. Most widely used compared to other storage classes	Used extensively for loop counters	Used extensively for recursive functions	Used in case of variables which are being used by almost all the functions in a program

Memory-related: memset(), memcpy(), memmove(), memscan(),
memcmp(), memchr()

memset():-

memset - fill memory with a constant byte

SYNOPSIS

```
      #include <string.h>
void *memset(void *s, int c, size_t n);
```

String-related: strcpy(), strcat(), strcmp(), strchr(),
strrchr(), strlen() and variants

Bitset:-

Bitset is very similar to vector<bool> (also known as bit_vector):

It contains a collection of bits, and provides constant-time access to each bit. There are two main differences between bitset and vector<bool>. First, the size of a bitset cannot be changed: bitset's template parameter N, which specifies the number of bits in the bitset, must be an integer constant. Second, bitset is not a Sequence; in fact, it is not an STL Container at all. It does not have iterators, for example, or begin() and end() member functions. Instead, bitset's interface resembles that of unsigned integers. It defines bitwise arithmetic operators such as &=, |=, and ^=. In general, bit 0 is the least significant bit and bit N-1is the most significant bit.

Example

```
int main() {
  const bitset<12> mask(2730ul);
  cout << "mask =      " << mask << endl;

  bitset<12> x;

  cout << "Enter a 12-bit bitset in binary: " << flush;
  if (cin >> x) {
    cout << "x =        " << x << endl;
    cout << "As ulong:  " << x.to_ulong() << endl;
    cout << "And with mask: " << (x & mask) << endl;
    cout << "Or with mask:  " << (x | mask) << endl;
  }
}
```

Calloc vs Malloc

Ans:-

There are two differences. First, is in the number of arguments.

Malloc() takes a single argument (memory required in bytes), while calloc() needs two arguments.

Secondly, malloc() does not initialize the memory allocated, while calloc() initializes the allocated memory to ZERO.

calloc() allocates a memory area, the length will be the product of its parameters. calloc fills the memory with ZERO's and returns a pointer to first byte. If it fails to locate enough space it returns a NULL pointer.

Syntax:

ptr_var=(cast_type *)calloc(no_of_blocks , size_of_each_block);

i.e. ptr_var=(type *)calloc(n,s);

malloc() allocates a single block of memory of REQUSTED SIZE and returns a pointer to first byte. If it fails to locate requsted amount of memory it returns a null pointer.

Syntax:

ptr_var=(cast_type *)malloc(Size_in_bytes);

1.What are the differences between structures and union?

A **structure** variable contains each of the named members, and its size is large enough to hold all the members. Structure elements are of same size.

A **Union** contains one of the named members at a given time and is large enough to hold the largest member. Union element can be of different sizes.

2.What is the difference between arrays and pointers?

Array is collection of similar datatype. it is a static memory allocation means we can not increment and decrement the arry size once we allocated. and we can not increment the base address, reassign address.

Pointer is a dynamic memory allocation. we can allocate the size as we want, assigning into another variable and base address incrementation is allowed.

What is the difference between array and pointer?

Array

Array allocates space automatically.

It cannot be resized.

It cannot be reassigned.

size of (arrayname) gives the number of bytes occupied by the array.

Pointer

Explicitly assigned to point to an allocated space.

It can be sized using realloc() 3-pointer can be reassigned.

sizeof (p) returns the number of bytes used to store the pointer variable p.

3.What is the difference between class and structure?

->By default, the members ot structures are public while that tor class is private.

->structures doesn’t provide something like data hiding which is provided by the classes.

->structures contains only data while class bind both data and member functions.

4.What is the difference between a string copy (strcpy) and a memory copy (memcpy)?

The strcpy() function is designed to work exclusively with strings. It copies each byte of the source string to the destination string and stops when the terminating null character () has been moved.

On the other hand, the memcpy() function is designed to work with any type of data. Because not all data ends with a null character, you must provide the memcpy() function with the number of bytes you want to copy from the source to the destination.

What is the difference between the functions memmove() and memcpy()?

The arguments of memmove() can overlap in memory. The arguments of memcpy() cannot.

5.What are the differences between new and malloc?

->New initializes the allocated memory by calling the constructor. Memory allocated with new ->should be released with delete.

->Malloc allocates uninitialized memory.

->The allocated memory has to be released with free.new automatically calls the constructor while malloc(dosen’t)

What is the difference between syntax vs logical error?

Syntax Error

->These involves validation of syntax of language.

->compiler prints diagnostic message.

Logical Error

->logical error are caused by an incorrect algorithm or by a statement

mistyped in such a way that it doesn’t violet syntax of language.

->difficult to find.

Data Structures Interview Questions with Answers

What is data structure ?
The logical and mathamatical model of a particular organization of data is called data structure.
Two types
-linear
-nonlinear

what is a linked list?
A linked list is a linear collection of data elements , called nodes , where the linear order is given by pointers.
Each node has two parts, first part contain the information of the element, second part contans the address of the next node in the list.

Define double linked list?
It is a collection of data elements called nodes, where each node is divided into three parts
- info field that contains the information stored inthe node.
-left field that contain pointer to node on left side.
-right field that contain pointer to node on right side.

What is a queue?
A queue is an ordered collection of items from which items may be deleted at one end (front end) and inserted at the other end (rear end).
It obeys FIFO rule there is no limit to the number of elements a queue contains .

What is difference between a Stack and an array?

STACK	ARRAY
-> Stack is a dynamic object whose size is constantly changing as item are pushed and poped	->array is ordered collection of items.
-> Stack may contain different data type.	-> array ontains same data types.
-> Stack is a ordered collection of items	-> Array is a static object.

C ++

Q.What is the difference between C and C++?

No.	C	C++
1)	C follows the procedural style programming.	C++ is multi-paradigm. It supports both procedural and object oriented.
2)	Data is less secured in C.	In C++, you can use modifiers for class

		members to make it inaccessible for outside users.
3)	C follows the top-down approach.	C++ follows the bottom-up approach.
4)	C does not support function overloading.	C++ supports function overloading.
5)	In C, you can't use functions in structure.	In C++, you can use functions in structure.
6)	C does not support reference variables.	C++ supports reference variables.
6)	In C, scanf() and printf() are mainly used for input/output.	C++ mainly uses stream cin and cout to perform input and output operations.

Q.What is the difference between reference and pointer?

No.	Reference	Pointer
1)	References are less powerful than pointers. Once a reference is created, it can't refer to other object later.	Pointers provide the powerful facilities than references.
2)	References are safer and easier to use than pointers.	Pointers are comparatively difficult to use.

Q.What is a class?

Class is a user-defined data type. Class defines the type definition of category of things. It defines a datatype, but it does not define the data it just specifies the structure of data.

You can create N number of objects from a class.

Q. What is an object?

Object is the instance of a class. A class provides a blueprint for objects. So you can create an object from a class. The objects of a class are declared with the same sort of declaration that we declare variables of basic types.

Q.What are the C++ access specifiers?

The access specifiers are used to define how to functions and variables can be accessed outside the class.

There are three types of access specifiers:

Q.What is Object Oriented Programming (OOP)?

OOP is a methodology or paradigm that provides many concepts. The basic concepts of Object Oriented Programming are given below:

Classes and Objects: Classes are used to specify the structure of the data. They define datatype. You can create any number of objects from a class. Objects are the instances of classes.

Encapsulation: Encapsulation is a mechanism which binds the data and associated operations together and thus hide the data from outside world. Encapsulation is also known as data hiding. In C++, It is achieved using the access specifiers i.e. public, private and protected .

Abstraction: Abstraction is used to hide the internal implementations and show only the necessary details to the outer world. Data abstraction is implemented using interfaces and abstract classes in C++.

Some people confused about Encapsulation and abstraction. But they both are different.

Inheritance: Inheritance is used to inherit the property of one class into another class. It facilitates you to define one class in term of another class.

Q. What is the difference between array and a list?

- Array is a collection of homogeneous elements while list is a collection of heterogeneous elements.
- Array memory allocation is static and continuous while List memory allocation is dynamic and random.
- In Array, users don't need to keep in track of next memory allocation while In list user has to keep in track of next location where memory is allocated.

Q. What is the difference between new() and malloc()?

- new() is a preprocessor while malloc() is a function.
- There is no need to allocate the memory while using "new" but in malloc() you have to use sizeof().
- "new" initializes the new memory to 0 while malloc() gives random value in the newly allotted memory location.

Q. What are the methods of exporting a function from a DLL?

There are two ways:

- By using the DLL's type library.
- Taking a reference to the function from the DLL instance.

Q.Define friend function.

Friend function acts as friend of the class. It can access the private and protected members of the class. The friend function is not a member of the class but it must be listed in the class definition.

Q. What is virtual function?

A virtual function is used to replace the implementation provided by the base class. The replacement is always called whenever the object in question is actually of the derived class, even if the object is accessed by a base pointer rather than a derived pointer.

Q. When should we use multiple inheritance?

You can answer this question in three manners:

- Never
- Rarely
- If you find that the problem domain cannot be accurately modeled any other way.

Q. What is the destructor?

Destructor is used to delete any extra resources allocated by the object.

Q. What is an overflow error?

It is a type of arithmetical error. It is happen when the result of an arithmetical operation been greater than the actual space provided by the system.

Q. What is overloading?

C++ facilitates you to specify more than one definition for a function name or an operator in the same scope. It is called function overloading and operator overloading respectively.

Q. What is function overriding?

If you inherit a class into a derived class and provide a definition for one of the base class's function again inside the derived class, then this function is called overridden function and this mechanism is known as function overriding.

Q. What is virtual inheritance?

Virtual inheritance facilitates you to create only one copy of each object even if the object appears more than one in the hierarchy.

Q. What is constructor?

Constructor is a special method that initializes object. It name must be same as class name.

Q. What is the purpose of "delete" operator?

The "delete" operator is used to release the dynamic memory created by "new" operator.

Q. Explain this pointer?

This pointer holds the address of current object.

Q. What does Scope Resolution operator?

A scope resolution operator(::) is used to define the member function outside the class.

Q. What is the difference between delete and delete[]?

Delete [] is used to release the array of allocated memory which was allocated using new[] whereas delete is used to release one chunk of memory which was allocated using new.

Q.Define the private, protected and public in C++?

Private: The data members and functions cannot be accessed from outside the class.

Protected: The data members and functions are accessible to derived class only.

Public: The data members and functions can be accessed from outside the class.

- **Private**: Functions and variables declared as private can be accessed only within the same class and they cannot be accessed outside the class they are declared.
- **Public**: Functions and variables declared under public can be accessed from anywhere.
- **Protected**: Functions and variables declared as protected cannot be accessed outside the class except a child class. This specifier is generally used in inheritance.

Data Structure Interview Questions

A list of top frequently asked **Data Structure interview questions** and answers are given below.

1) What is Data Structure? Explain.

Data structure is a way that specifies how to organize and manipulate the data. It also specifies the relationship between them. It provides some algorithms to make data access more efficient and easy.

2) In which areas data structures are applied extensively?

Data structures are applied extensively in the following areas of computer science:

- Compiler Design,
- Operating System,
- Database Management System,
- Statistical analysis package,
- Numerical Analysis,
- Graphics,
- Artificial Intelligence,
- Simulation

3) What is the difference between file structure and storage structure?

Difference between file structure and storage structure:

The main difference between file structure and storage structure is based on memory area that is being accessed.

Storage structure: When we deal with the structure that resides in the main memory of the computer system, known as the storage structure.

File structure: When we deal with an auxiliary structure then it is referred as file structures.

4) Which data structures are used with the following areas: RDBMS, Network data model and hierarchical data model?

- RDBMS uses Array data structure
- Network data model uses Graph
- Hierarchal data model uses Trees

5) What are Binary trees?

A Binary Tree is a type of data structure that has two nodes: A left node and a right node. In programming, binary trees are actually an extension of the linked list structures.

6) What is a Stack?

A stack is a data structure in which only the top element can be accessed. As data is stored in the stack, each data is pushed downward, leaving the most recently added data on top.

7) What is a multidimensional array?

A multidimensional array stores data in multiple indexes. It is used when the storing data that cannot be represented using a single dimensional indexing, such as data representation in a board game, tables with data stored in more than one column.

8) What is a linked list in data structure?

A linked list is a sequence of nodes in which each node is connected to the node following it. It makes a chain like link of data storage.

9) If you are using C language to implement the heterogeneous linked list, what pointer type should be used?

The heterogeneous linked list contains different data types, so it is not possible to use ordinary pointers for this. For this work, you have to use a generic pointer type like void pointer because void pointer is capable of storing pointer to any type.

10) How many minimum numbers of queues are needed to implement the priority queue?

Two queues are needed. One queue is used for actual storing of data and another for storing priorities.

11) Which data structure is used to perform recursion?

Stack is used to perform recursion because of its LIFO (Last In First Out) property. It knows whom to return when the function has to return.

12) When should you use binary search engine?

A binary search algorithm is used to search a list when the elements are already in order or sorted. The list starts searching in the middle, if the middle value is not the target search key, it will check to see if it will continue the search on the lower half of the list or the higher half. The split and search will then continue in the same manner.

13) How to reference all the elements in a one-dimension array?

It can be done by using an indexed loop such that the counter runs from 0 to the array size minus one. By this manner, you can reference all the elements in sequence by using the loop counter as the array subscript.

14) Which notations are used in Evaluation of Arithmetic Expressions using prefix and postfix forms?

Polish and Reverse Polish notations.

15) Give some example of the application of Tree-data structure?

Application of Tree- data structure:

- The manipulation of Arithmetic expression,
- Symbol Table construction,
- Syntax analysis

16) Give the example of some applications that make use of Multilinked Structures?

- Sparse matrix,
- Index generation.

17) Are linked lists considered linear or non-linear data structures?

A linked list is considered both linear and non-linear data structure depending on the situation.

- On the basis of data storage, it is considered as non-linear data structure.
- On the basis of access strategy, it is considered as linear data-structure.

18) What is the difference between NULL and VOID?

- Null is actually a value, whereas Void is a data type identifier.

- A null variable simply indicates an empty value, whereas void is used to identify pointers as having no initial size.

19) What is the difference between PUSH and POP?

PUSH and POP operations specify how data is stored and retrieved in a stack.

PUSH: PUSH specifies that data is being "pushed" into the stack.

POP: POP specifies data retrieval, and in particular refers to the topmost data being accessed.

20) What is a postfix expression?

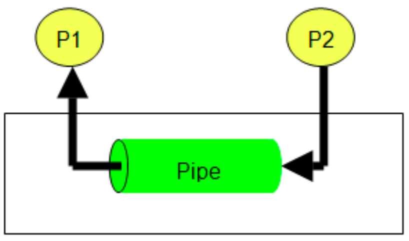
An expression which each operator follows its operand is known as postfix expression. The main benefit of this form is that there is no need to group sub-expressions in parentheses or to consider operator precedence.

Linux

Note: *Pipes are used for communication between related processes(parent-child-sibling) on the same linux machine.*

#include <unistd.h>
int pipe(int file_descriptor[2]);

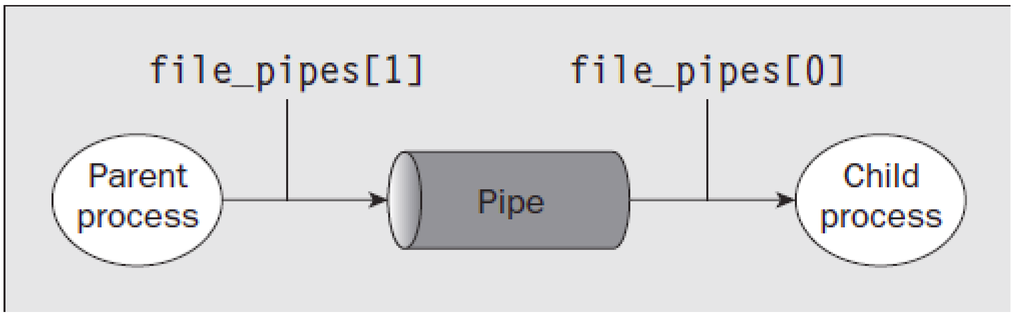
The two file descriptors returned are connected in a special way. Any data written to file_descriptor[1] can be read back from file_descriptor[0]. The data is processed in a first in, first out basis, usually abbreviated to FIFO. This means that if you write the bytes 1, 2, 3 to file_descriptor[1], reading from file_descriptor[0] will produce 1, 2, 3.
This is different from a stack, which operates on a last in, first out basis, usually abbreviated to LIFO.



```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
```

```
int main()
{
        int res,fd[2],s;
        char str[25],str1[25];
        res=pipe(fd);
        if(res==-1)
        {
                perror("\nPipe Creation Failed...\n");
                exit(1);
        }
        else
        {
                printf("\nPipe Created Successfully...\n");
                printf("\nDescriptors are:%d\t%d\n",fd[0],fd[1]);
                switch(fork())
                {
                        case -1:
                                perror("\nFork Failed....\n");
                                exit(1);
                        case 0:
                                wait(&s);
                                res=read(fd[0],str1,sizeof(str1));
                                if(res<0)
                                {
                                        perror("\nRead Error...\n");
                                        exit(1);
                                }
                                else
                                {
                                        printf("\nChild Process\n");
                                        printf("\nPipe Read Successfully\n");
                                        printf("\nData:%s\n",str1);
                                }
                                break;
                        default:
                                printf("\nParent Process....\n");
                                printf("\nEnter Data To Be Written To Pipe\n");
                                gets(str);
                                res=write(fd[1],str,sizeof(str));
                                if(res<0)
                                {
                                        perror("\nWrite Error Occurred\n");
                                        exit(1);
                                }
                                else
```

```
                                        {
                                                  printf("\nPipe Written Successfully..\n");
                                        }
                                        break;
                    }
          }
}
```



Named PIPE (FIFO’s) :

Drawbacks of PIPE :

1.In PIPE communication, only intra-communication between processes is possible i.e. related processes only.

2.Scope of the PIPE is temporary i.e. only upto the program is running.

3.We have to handle file descriptors for communication between two processes.

So, the next inter process communication mechanism is Named PIPE (FIFO).

You can create named pipes from the command line and from within a program.

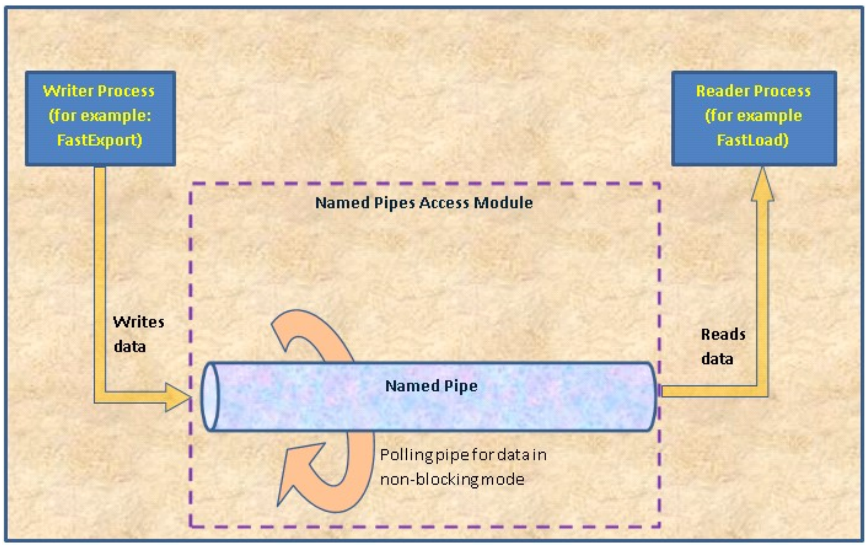
The preferred command-line method is to use-

```
# mkfifo filename
```

From inside a program, you can use two different calls:

```
#include <sys/types.h>
#include <sys/stat.h>
int mkfifo(const char *filename, mode_t mode);
```

Note: Named Pipes(FIFO) are used for communication between related and unrelated processes on the sane LINUX machine.


Writer Process (for example: FastExport)
Reader Process (for example FastLoad)
Named Pipes Access Module
Writes data
Reads data
Named Pipe
Polling pipe for data in non-blocking mode

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>

int main()
{
        int res,fd;
        char str[50],str1[50];
        if(access("my_fifo",F_OK)==-1)
        {
                printf("\nFile Permission Are OK\n");
                res=mkfifo("my_fifo",0777);
                if(res==-1)
                {
                        perror("\nFifo Can't Created\n");
                        exit(1);
                }
        }
        printf("\nFifo Created Successfully...\n");
        fd=open("my_fifo",O_RDWR);
        if(fd==-1)
        {
                perror("\nFifo Can't Open..\n");
                exit(1);
        }
        else
```

```
	{
		printf("\nEnter Data To Be Written..\n");
		//scanf("%s",str);
		gets(str);
		res=write(fd,str,sizeof(str));
		if(res<0)
		{
			perror("\nWrite Error..\n");
			exit(1);
		}
		else
		{
			printf("\nWritten Successfully.\n");
		}
		lseek(fd,0,SEEK_SET);
		res=read(fd,str1,sizeof(str1));
		if(res<0)
		{
			printf("\nRead Error....\n");
			exit(1);
		}
		else
		{
			printf("\nData From Named Pipe Is:%s\n",str1);
		}
	}
}
```

Message Queue :

<u>Drawbacks of Named-PIPE</u> :

1. In named pipe, communication is only possible if there is a receiver at the opposite to sender side. i.e. receiver must be present to receive message otherwise message will be lost.

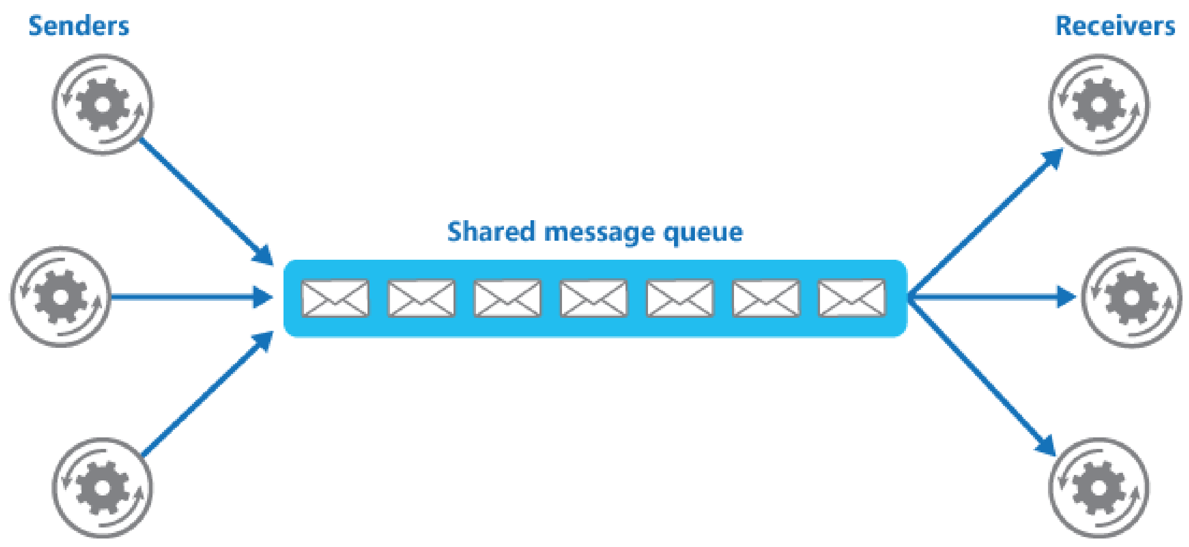
So, the next inter process communication mechanism is Message Queue.

<u>Advantages to use Message Queue :</u>

1. If the receiver is not present even then also the sender can send a message and the messages will be received when the receiver is present or online we can say in FIFO manner. i.e. the message send first will be received first.

2. Communication is very fast using message queue. Client-Serve can be implemented easily.

Message queues provide a way of sending a block of data from one process to another. Message queues provide a reasonably easy and efficient way of passing data between two unrelated processes. They have the advantage over named pipes that the message queue exists independently of both the sending and receiving processes, which removes some of the difficulties that occur in synchronizing the opening and closing of named pipes.



Message Queue Functions :
#include <sys/msg.h>
int msgctl(int msqid, int cmd, struct msqid_ds *buff);
int msgget(key_t key, int msgflg);
int msgrcv(int msqid, void *msg_ptr, size_t msg_sz, long int msgtype, int msgflg);
int msgsnd(int msqid, const void *msg_ptr, size_t msg_sz, int msgflg);

<u>msgget :</u>
This function is used to create a message queue and get a key to access this message queue.

Syntax :

# int msgget(key_t key, int msgflg);

A special bit defined by IPC_CREAT must be bitwise ORed with the permissions to create a new message queue. The IPC_CREAT flag is silently ignored if the message queue already exists. The msgget function returns a positive number, the queue identifier, on success or –1 on failure.
<u>msgsnd :</u>

The **msgsnd** function allows you to add a message to a message queue:

Syntax :

# int msgsnd(int msqid, const void *msg_ptr, size_t msg_sz, int msgflg);

- The first parameter, msqid, is the message queue identifier returned from a msgget function.
- The second parameter, msg_ptr, is a pointer to the message to be sent, which must start with a long int type as described previously.
- The third parameter, msg_sz, is the size of the message pointed to by msg_ptr. This size must not include the long int message type.
- The fourth parameter, msgflg, controls what happens if either the current message queue is full or the systemwide limit on queued messages has been reached. If msgflg has the IPC_NOWAIT flag set, the function will return immediately without sending the message and the return value will be –1.
  If the msgflg has the IPC_NOWAIT flag clear, the sending process will be suspended, waiting for space to become available in the queue.
- On success, the function returns 0, on failure –1. If the call is successful, a copy of the message data has been taken and placed on the message queue.

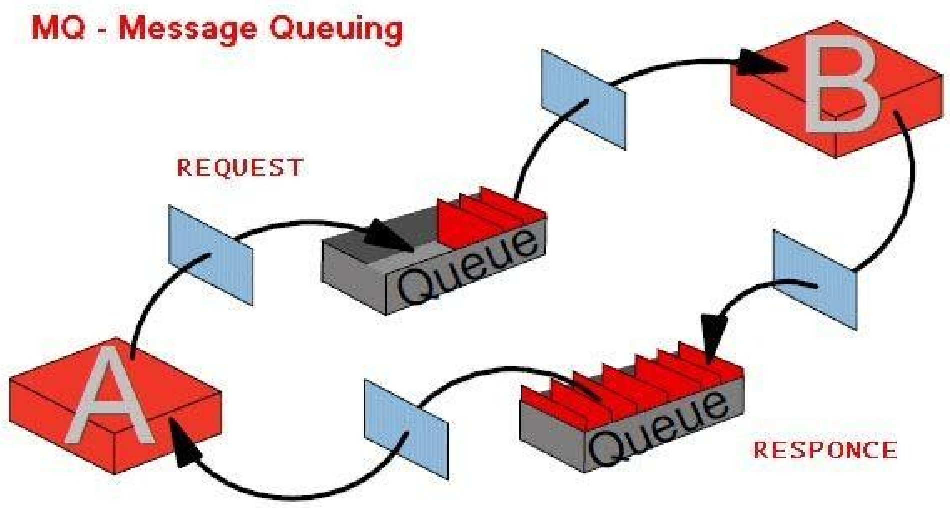
msgrcv :

The msgrcv function retrieves messages from a message queue:

Syntax :

# int msgrcv(int msqid, void *msg_ptr, size_t msg_sz, long int msgtype, int msgflg);

- The first parameter, msqid, is the message queue identifier returned from a msgget function.
- The second parameter, msg_ptr, is a pointer to the message to be received, which must start with a long int type as described previously in the msgsnd function.
- The third parameter, msg_sz, is the size of the message pointed to by msg_ptr, not including the long int message type.
- The fourth parameter, msgtype, is a long int, which allows a simple form of reception priority to be implemented.



```
#include<stdio.h>
#include<sys/ipc.h>
#include<sys/types.h>
#include<sys/msg.h>
#include<stdlib.h>
#include<unistd.h>
#include<string.h>

struct msgqbuf
{
        long mtype;
        char mtext[50];
};
typedef struct msgqbuf msgque;
int main()
{
        int msgqid,siz,siz1,i;
        char ch;
        msgque q,q1;
        msgqid=msgget(1234,0666|IPC_CREAT);
        if(msgqid==-1)
        {
                perror("\nMsg Queue Can't Create...\n");
                exit(1);
        }
        else
        {
                printf("\nMsg Queue Created Successfully...\n");
                q.mtype=0;
                do
                {
                        printf("\nEnter Data To Be sent\n");
                        ch=getchar();
                      gets(q.mtext);
                        siz=msgsnd(msgqid,q.mtext,strlen(q.mtext),0);
                        if(siz==-1)
                        {
                                perror("\nMsg Sending Failed....\n");
                                exit(1);
                        }
                        else if(siz==0)
                        {
                                printf("\nMsg Sent Successfully...\n");
                        }
                printf("Do u want to continue sending msg..\nY:Yes\nN:No\n");
                scanf(" %c",&ch);
                }while(ch=='Y'||ch=='y');
        }
}
```

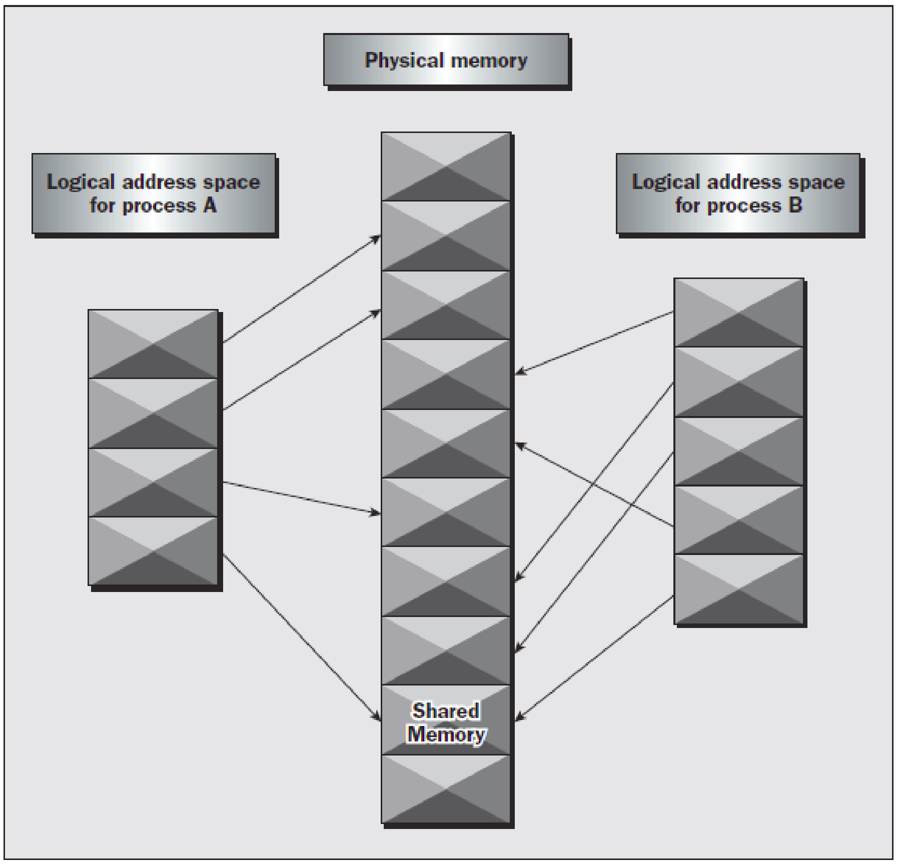
Disadvantage of Message Queue :-

1. The only disadvantage of message queue is that we can send data only of 1024 bytes. But in shared memory the more size of data can be transfer between two processes and that depends on the physical memory of the system.

So to overcome this problem we use shared memory concept.

Shared Memory :

In this concept two unrelated processes can access the same logical memory. Shared memory is a very efficient way of transferring data between two running processes. Shared memory is a special range of addresses that is created by IPC for one process and appears in the address space of that process. Other processes can then “attach” the same shared memory segment into their own address space. All processes can access the memory locations just as if the memory had been allocated by malloc. If one process writes to the shared memory, the changes immediately become visible to any other process that has access to the same shared memory.


Physical memory
Logical address space for process A
Logical address space for process B
Shared Memory

```
#include <sys/shm.h>
void *shmat(int shm_id, const void *shm_addr, int shmflg);
int shmctl(int shm_id, int cmd, struct shmid_ds *buf);
int shmdt(const void *shm_addr);
int shmget(key_t key, size_t size, int shmflg);
```

shmget :

This function is defined in sys/shm.h header file and used to create shared memory.

Syntax :

```
# int shmget(key_t key, size_t size, int shm_flag);
```

```
1  # int shmget(key_t key, size_t size, int shm_flag);
```

It returns a shared memory identifier and flay used is IPC_PRIVATE that creates a private shared memory to the processes and IPC_CREAT flag to create memory. The second argument is size which tells the amount of memory required for processes in bytes.

shmat :

To enable access to the memory that is created we have to attach it to the address space of a process, so this is done with the help of shmat function.

Syntax :

```
# void *shmat(int shm_id, const void *shm_addr, int shm_flag);
```

shmdt:

The shmdt function detaches the shared memory from the current process. It takes a pointer to the address returned by shmat. On success, it returns 0, on error –1.

EXAMPLE :

First at the sender side we will create a logical memory by using shmget function of size say 1024 bytes.

sender.c –

```
#include<sys/ipc.h>
#include<sys/shm.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int main()
{
        int shmid,run=1;
        char *shm_mem,str[50],i,ch;
        void *ptr;
        shmid=shmget(1234,1024,0666|IPC_CREAT);
        if(shmid==-1)
        {
                perror("\nShared Memmory Can't Be Allocated....\n");
                exit(1);
```

```
	}
	else
	{
		printf("\nShared Memory Allocated Successfully...\n");
		ptr=shmat(shmid,(void *)0,0);
		if(ptr==(void *)-1)
		{
			perror("\nMemory Attachment Failed..\n");
			exit(1);
		}
		else
		{
			printf("\nMemory Attached Successfully At Add:%ld\n",(int
*)ptr);
			while(run)
			{
				i=0;
				ch=getchar();
				do
				{
					str[i]=ch;
					ch=getchar();
					i++;
				}while(ch!='\n');
				str[i]='\0';
				if(strcmp(str,"bye")==0)
				{
					run=0;
				}
				else
				{
					shm_mem=(char *)ptr;
					strcpy(shm_mem,str);
					printf("\nData U written :%s\n",shm_mem);
					sleep(2);
				}

			}
		}
	}
}
```

OUTPUT :

Here the shared memory is created to communication between two processes and the memory address is returned by a pointer. Second process is say receiver.c that will receive the data from that shared memory.

receiver.c –

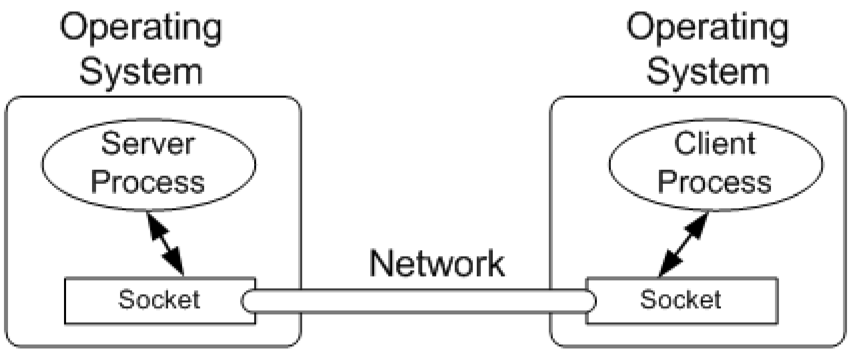
```
#include<sys/ipc.h>
#include<sys/shm.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int main()
{
        int shmid,run=1;
        char *shm_mem,str[50],i,ch;
        void *ptr;
        shmid=shmget(1234,1024,0666|IPC_CREAT);
        if(shmid==-1)
        {
                perror("\nShared Memmory Can't Be Allocated....\n");
                exit(1);
        }
        else
        {
                printf("\nShared Memory Allocated Successfully...\n");
                ptr=shmat(shmid,(void *)0,0);
                if(ptr==(void *)-1)
                {
                        perror("\nMemory Attachment Failed..\n");
                        exit(1);
                }
                else
                {
                        printf("\nMemory Attached Successfully At Add:%ld\n",(int
*)ptr);
                        shm_mem=(char *)ptr;
                        strcpy(str,shm_mem);
                while(1)
       {
                        if(run)
                        {
                                printf("\nData U written :%s\n",str);
                                sleep(1);
                                run=0;
                        }
                        else if(str=="bye")
                        {
                                run=0;
                                exit(0);
                        }
        }

                }
          }
}
```

Disadvantage of Shared Memory :

By itself, shared memory doesn't provide any synchronization facilities. So you usually need to use some other mechanism to synchronize access to the shared memory. Typically, you might use shared memory to provide efficient access to large areas of memory and pass small messages to synchronize access to that memory. There are no automatic facilities to prevent a second process from starting to read the shared memory before the first process has finished writing to it. It's the responsibility of the programmer to synchronize access.

<u>SOCKETS</u> :

A socket is a communication mechanism that allows client/server systems to be developed either locally, on a single machine, or across networks. Linux functions such as printing, connecting to databases, and serving web pages as well as network utilities such as rlogin for remote login, http for hyper text transfer protocol and ftp for file transfer usually use sockets to communicate. Sockets are created and used differently from pipes because they make a clear distinction between client and server. The socket mechanism can implement multiple clients attached to a single server.



<u>Socket Connections</u>

First, a server application creates a socket, which like a file descriptor is a resource assigned to the server process and that process alone. The server creates it using the system call socket, and it can't be shared with other processes.

Next, the server process gives the socket a name. For example, a web server typically creates a socket on port 80, an identifier reserved for the purpose. Web browsers know to use port 80 for their HTTP connections and port 443 for HTTPS connections.

- A socket is named using the system call bind.
- The system call, listen, creates a queue for incoming connections.
- The server can accept them using the system call accept.

When the server calls accept, a new socket is created that is distinct from the named socket. This new socket is used solely for communication with this particular client.

The client side of a socket-based system is more straightforward. The client creates an unnamed socket by calling socket.

Socket Attributes

Sockets are characterized by three attributes: domain, type, and protocol.

Socket Domains : Domains specify the network medium that the socket communication will use. The most common socket domain is AF_INET, which refers to Internet networking that's used on many Linux local area networks and, of course, the Internet itself.

Protocols : Internet protocols provide two communication mechanisms with distinct levels of service: streams and datagrams.

- **Stream Sockets :** Stream sockets (in some ways similar to standard input/output streams) provide a connection that is a sequenced and reliable two-way byte stream. Thus, data sent is guaranteed not to be lost, duplicated, or
reordered without an indication that an error has occurred.
- **Datagram Sockets :** A datagram socket, specified by the type SOCK_DGRAM, doesn't establish and maintain a connection. There is also a limit on the size of a datagram that can be sent. It's transmitted as a single network message that may get lost, duplicated, or arrive out of sequence.

Creating a Socket

The socket system call creates a socket and returns a descriptor that can be used for accessing the socket.

Syntax :

#include <sys/types.h> #include <sys/socket.h> int socket(int domain, int type, int protocol);

```
#include <sys/types.h>
#include
<sys/socket.h>
int socket(int domain,
int type, int protocol);
```

Naming a Socket

To make a socket (as created by a call to socket) available for use by other processes, a server program needs to give the socket a name. The bind system call assigns the address specified in the parameter, address, to the unnamed socket associated with the file descriptor socket. The length of the address structure is passed as address_len.

Syntax :

```
#include <sys/socket.h> int bind(int socket, const struct sockaddr *address, size_t address_len);
```

```
#include <sys/socket.h>
int bind(int socket, const
struct sockaddr
*address, size_t
address_len);
```

Creating a Socket Queue

To accept incoming connections on a socket, a server program must create a queue to store pending requests. It does this using the listen system call.

Syntax :

```
#include <sys/socket.h> int listen(int socket, int backlog);
```

```
#include <sys/socket.h>
int listen(int socket, int
backlog);
```

A value of 5 for backlog is very common.

Accepting Connections

Once a server program has created and named a socket, it can wait for connections to be made to the socket by using the accept system call.

Syntax :

```
#include <sys/socket.h> int accept(int socket, struct sockaddr *address, size_t
*address_len);
```

Requesting Connections

Client programs connect to servers by establishing a connection between an unnamed socket and the server listen socket. They do this by calling connect.

Syntax :

```
#include <sys/socket.h>
int connect(int socket, const struct sockaddr *address, size_t address_len);
```

Closing a Socket

You can terminate a socket connection at the server and client by calling close, just as you would for low level file descriptors.

Example of SOCKET Communication in LINUX

Let us take a server-client example to illustrate socket communication between two processes.

- **client.c**

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<sys/types.h>
```

```
#include<sys/socket.h>
#include<netinet/in.h>
#include<netdb.h>
#include<unistd.h>
#define DATA "Hello World"

int main(int argc,char *argv[])
{
	int sock,i;
	struct hostent *hp;
	struct sockaddr_in server;
	char buff[1024],data[100],ch;
	while(1)
	{
	sock=socket(AF_INET,SOCK_STREAM,0);
	if(sock<0)
	{
		perror("\nSocket failed\n");
		exit(1);
	}
	server.sin_family=AF_INET;
	hp=gethostbyname(argv[1]);
	if(hp==0)
	{
		perror("\ngethostbyname failed......\n");
		exit(1);
	}
	memcpy(&server.sin_addr,hp->h_addr,hp->h_length);
	server.sin_port=htons(6000);		//converts unsigned short integer from
host byte order to network bbyte order.
	if(connect(sock,(struct sockaddr *)&server,sizeof(server))<0)
	{
		perror("Connect failed\n");
		exit(1);
	}
	printf("Enter data to send:");
//	scanf("%s",data);
	i=0;
	ch=getchar();
	do
	{
		data[i]=ch;
		ch=getchar();
		i++;
	}while(ch!='\n');
```

```
        data[i]='\0';
        if(send(sock,data,sizeof(data),0)<0)
        {
                perror("send failed\n");
                close(sock);
                exit(1);
        }
        printf("\nSent:%s\n",data);
        close(sock);
        }
        return 0;

}
```

On compiling client.c, we provide localhost as a command line argument to connect the system with localhost i.e. 127.0.0.1, But the connection was refused because the server is unavaliable.

server.c

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<string.h>
#include<stdlib.h>

int main(int argc,char *agrv[])
{
        /*Variables*/
        int sock;
        struct sockaddr_in server;
        int mysock;
        char buff[1024];
        int rval;

        /*Create Socket*/

        sock=socket(AF_INET,SOCK_STREAM,0);
        if(sock<0)
        {
                perror("\nFailed to create socket\n");
                exit(1);
        }
        server.sin_family=AF_INET;
        server.sin_addr.s_addr=INADDR_ANY;
        server.sin_port=htons(6000);

        /*Call Bind*/

        if(bind(sock,(struct sockaddr *)&server,sizeof(server)))
        {
                perror("\nBind Failed\n");
                exit(1);
        }

        /*Listen*/

        listen(sock,5);

        /*Accept*/

        do
        {
                printf("\nWaiting....\n");
                mysock=accept(sock,(struct sockaddr *)0,0);
                if(mysock==-1)
                {
                        perror("\nAccept failed\n");
                        exit(1);
                }
                else
                {
                        memset(buff,0,sizeof(buff));
                        if((rval=recv(mysock,buff,sizeof(buff),0))<0)
                        {
                                perror("\nReading stream message error\n");
                                exit(1);
                        }
                        else if(rval==0)
                                printf("\nEnding Connection\n");
                        else
                        {
                                printf("MSG:%s\n",buff);
                        }
                        printf("Got the message(rval=%d)",rval);
                        close(mysock);
                }
        }while(1);
        return 0;
```

}

Q.What is CLI?
CLI stands for Command Line Interface. It is an interface that allows users to type declarative commands to instruct the computer to perform operations.

Q.What is GUI?

GUI stands for Graphical User Interface. It uses the images and the icons which are clicked by the users to communicate with the system. It is more attractive and user-friendly because the use of the images and icons.

Q.What is the basic difference between BASH and DOS?

- BASH commands are case sensitive while DOS commands are not case sensitive.
- DOS follows a convention in naming files. In DOS, 8 character file name is followed by a dot and 3 characters for the extension. BASH doesn't follow such convention.

Q.What is the maximum length for a file name in Linux?
255 characters.

Q.What is a gateway? Is there any difference between gateway and router?

A node that is connected to two or more networks is commonly known as gateway. It is also known as router. It is used to forward messages from one network to another.

Q.What is DNS?

DNS is an acronym stands for Domain Name System. It is a naming system for all the resources over internet which includes physical nodes and applications. It is used to locate to a resource easily over a network.

Q. What is RIP?

RIP stands for Routing information Protocol. It is accessed by the routers to send data from one network to another.

Q. What do you understand by TCP/IP?

TCP/IP is short for Transmission Control Protocol /Internet protocol. It is a set of protocol layers that is designed for exchanging data on different types of networks.

Q. What do you understand by ping command?

The "ping" is a utility program that allows you to check the connectivity between the network devices. You can ping devices using its IP address or name.

INNOMINDS EXPERIENCE

1ST ROUND:

Written test at the company office.

APTITUDE N VERBAL TEST: 1 hour

C: 1 hour (objective)

Topics to prepare:

1. **Aptitude**: profit and loss, problems on ages, partnership, time and work, odd one out and series.
2. **Verbal**: synonyms and antonyms, analogy, basic English grammar.
3. **C:** basic c programs …Reference-: Test ur C skills.

2nd ROUND:

TECHNICAL:

1. What is a structure? Why do we need them?

Ans:-

C Structure is a collection of different data types which are grouped together and each element in a C structure is called member.

C Structures can be used to store huge data. Structures act as a database.

1.C Structures can be used to send data to the printer.
2.C Structures can be used to clear output screen contents.
3.C Structures can be used to check computer's memory size etc.

2. Definition of Array and its uses.

Ans:-

C Array is a collection of variables belongings to the same data type. You can store group of data of same data type in an array.

Array might be belonging to any of the data types

Array size must be a constant value.

Always, Contiguous (adjacent) memory locations are used to store array elements in memory.

It is a best practice to initialize an array to zero or null while declaring, if we don't assign any values to array.

Multidimensional arrays

Arrays can themselves be members of arrays.

3. Linked lists…WHY? WHERE? And HOW are they useful?

Ans:-
Single Linked lists?

A linked list is a sequence of data structures, which are connected together via links. Linked List is a sequence of links which contains items. Each link contains a connection to another link. Linked list is the second most-used data structure after array.

Each node has two parts, first part contain the information of the element, second part contans the address of the next node in the list.

double linked list?

It is a collection of data elements called nodes, where each node is divided into three parts
- info field that contains the information stored inthe node.
-left field that contain pointer to node on left side.
-right field that contain pointer to node on right side.

4. What is a class? Why classes are required?

Ans:-

In object-oriented programming, a **class** is an extensible program-code-template for creating objects, providing initial values for state (member variables) and implementations of behavior (member functions or methods).

5. **Difference between class and structure.**

Ans;-

Class can create a subclass that will inherit parent's properties and methods, whereas Structure does not support the inheritance.

A class has all members private by default. A struct is a class where members are public by default.

Sizeof empty class is 1 Byte where as Sizeof empty structure is 0 Bytes

Classes are still fit for larger or complex objects and Structs are good for small, isolated model objects.

6. Use of static keyword in C for Variable declaration.

Ans:-

static variables are those variables whose life time remains equal to the life time of the program. Any local or global variable can be made static depending upon what the logic expects out of that variable.

7. How are static variables declared in c++?

Ans:-
1.Static member functions do not have this pointer.
2.A static member function cannot be virtual

```
#include<iostream>
class Test {
  static Test * fun() {
    return this; // compiler error
  }
```

};

3.Member function declarations with the same name and the name parameter-type-list cannot be overloaded if any of them is a static member function declaration.
Eg:- following program fails in compilation with error "'void Test::fun()' and `static void Test::fun()' cannot be overloaded"

```
#include<iostream>
class Test {
   static void fun() {}
   void fun() {} // compiler error
};
```

4.A static member function can not be declared *const*, *volatile*, or *const volatile*.

```
#include<iostream>
class Test {
   static void fun() const { // compiler error
     return;
   }
};
```

8. Difference between array of objects and variable array.
9. Reversing a linked list -> with recursion and without recursion.
10. difference between c and c++
Ans:-

C	**C++**
When compared to C++, C is a subset of C++.	C++ is a superset of C. C++ can run most of C code while C cannot run C++ code.
C supports procedural programming paradigm for code development.	C++ supports both procedural and object oriented programming paradigms; therefore C++ is also called a hybrid language.
C does not support function and operator overloading.	C++ supports both function and operator overloading.
C does not allow functions to be defined inside structures.	In C++, functions can be used inside a structure.
C does not have namespace feature.	C++ uses NAMESPACE which avoid name collisions.

C uses functions for input/output. For example scanf and printf.	**C++ uses objects** for input output. For example cin and cout.
C does not support reference variables.	C++ supports reference variables.
C has no support for virtual and friend functions.	C++ supports virtual and friend functions.
C provides malloc() and calloc() functions for dynamic memory allocation, and free() for memory de-allocation.	C++ provides new operator for memory allocation and delete operator for memory de-allocation.
C does not provide direct support for error handling (also called exception handling)	C++ provides support for exception handling. Exceptions are used for "hard" errors that make the code incorrect.

11. What is debugging, segmentation fault and stack overflow.
Ans:-

Debugging

Debugging is the routine process of locating and removing computer program bugs, errors or abnormalities, which is methodically handled by software programmers via debugging tools. Debugging checks, detects and corrects errors or bugs to allow proper program operation according to set specifications.

Debugging is also known as debug.

Segmentation fault

"Segmentation fault" means that you tried to access memory that you do not have access to.

stack overflow

A stack overflow is an undesirable condition in which a particular computer program tries to use more memory space than the call stack has available. In programming, the call stack is a buffer that stores requests that need to be handled.

12. What are recursive functions?
Ans:-

A recursive function (DEF) is a function which either calls itself or is in a potential cycle of function calls. As the definition specifies, there are two types of recursive functions. Consider a function which calls itself: we call this type of recursion immediate recursion.

13. Define process, thread. Difference between them.
Ans:-

Process:

A process is an instance of a program running in a computer.
A process is a program which is an exicution.

Thread:

thread is a light wait processes,

A thread is a single exicution stream in a process which can be indipendently sheduled by the kernel and which shares the address space.

The key differences

Threads	**Processes**
● Will by default share memory	● Will by default not share memory
● Will share file descriptors	● Most file descriptors not shared
● Will share filesystem context	● Don't share filesystem context
● Will share signal handling	● Don't share signal handling

Threads Vs. Processes

->All threads in a program must run the same executable. A child process, on the other hand, may run a different executable by calling an exec function.

->One thread can write the memory used by other thread |-> One process cannot corrupt another |

->Communication can be done by sharing global variables. |->Communication between processes | can be done using message passing

->Threads within a process share memory |->Don't share memory

->Threads are scheduled by OS |->Processes are not scheduled by OS

->Threads are active. |->Process are passive.

14. **What is fork, pthread_create?**

Ans:-

-> System call **fork()** is used to create processes. It takes no arguments and returns a process ID. The purpose of **fork()** is to create a ***new*** process, which becomes the *child* process of the caller. After a new child process is created, ***both*** processes will execute the next instruction following the ***fork()*** system call. Therefore, we have to distinguish the parent from the child. This can be done by testing the returned value of **fork()**:

- •If **fork()** returns a negative value, the creation of a child process was unsuccessful.
- •**Fork()** returns a zero to the newly created child process.
- •**Fork()** returns a positive value, the ***process ID*** of the child process, to the parent. The returned process ID is of type **pid_t** defined in **sys/types.h**.

Normally, the process ID is an integer. Moreover, a process can use function **getpid()** to retrieve the process ID assigned to this process.

-> The *pthread_create()* function is used to create a new thread

```
#include <pthread.h>

int pthread_create(pthread_t *thread, const pthread_attr_t *attr,void
*(*start_routine)(void*), void *arg);
```

3rd ROUND:
MANAGERIAL ROUND:
1. what is a browser?

A **browser** is an application program that provides a way to look at and interact with all the information on the World Wide Web. The word "**browser**" seems to have originated prior to the Web as a generic term for user interfaces that let you browse (navigate through and read) text files online.

2. difference between TCPIP and OSI model
Ans:-

OSI(Open System Interconnection)	**TCP/IP(Transmission Control Protocol / Internet Protocol)**
1. OSI is a generic, protocol independent standard, acting as a communication gateway between the network and end user.	1. TCP/IP model is based on standard protocols around which the Internet has developed. It is a communication protocol, which allows connection of hosts over a network.
2. In OSI model the transport layer guarantees the delivery of packets.	2. In TCP/IP model the transport layer does not guarantees delivery of packets. Still the TCP/IP model is more reliable.
3. Follows vertical approach.	3. Follows horizontal approach.
4. OSI model has a separate Presentation layer and Session layer.	4. TCP/IP does not have a separate Presentation layer or Session layer.
5. OSI is a reference model around which the networks are built. Generally it is used as a guidance tool.	5. TCP/IP model is, in a way implementation of the OSI model.
6. Network layer of OSI model provides both connection oriented and connectionless service.	6. The Network layer in TCP/IP model provides connectionless service.

7. OSI model has a problem of fitting the protocols into the model.	7. TCP/IP model does not fit any protocol
8. Protocols are hidden in OSI model and are easily replaced as the technology changes.	8. In TCP/IP replacing protocol is not easy.
9. OSI model defines services, interfaces and protocols very clearly and makes clear distinction between them. It is protocol independent.	9. In TCP/IP, services, interfaces and protocols are not clearly separated. It is also protocol dependent.
10. It has 7 layers	10. It has 4 layers

3. What is HTTP?

Ans:

HTTP (Hypertext Transfer Protocol) is the set of rules for transferring files (text, graphic images, sound, video, and other multimedia files) on the World Wide Web. As soon as a Web user opens their Web browser, the user is indirectly making use of **HTTP**.

4. explain the Internet layer in TCPIP model.

The **internet layer** is a group of internetworking methods, protocols, and specifications in the Internet protocol suite that are used to transport datagrams (packets) from the originating host across network boundaries, if necessary, to the destination host specified by a network address (IP address) which is defined for this purpose by the Internet Protocol (IP). The internet layer derives its name from its function of forming an internet (uncapitalized), or facilitating internetworking, which is the concept of connecting multiple networks with each other through gateways.

5. What is const volatile? Can we use it?

Ans:-

Volatile is key word used to prevent compiler to optimize a variable which can change unexpectedly beyound compilers comprehension.

6. How do we declare constant pointer to variable, pointer to constant, constant pointer to constant variable.

Constant Pointers

Lets first understand what a constant pointer is. A constant pointer is a pointer that cannot change the address its holding. In other words, we can say that once a constant pointer points to a variable then it cannot point to any other variable.

```
<type of pointer> * const <name of pointer>
int * const ptr;
```

Pointer to Constant

As evident from the name, a pointer through which one cannot change the value of variable it points is known as a pointer to constant. These type of pointers can change the address they point to but cannot change the value kept at those address.

```
const <type of pointer>* <name of pointer>
const int* ptr;
```

Constant Pointer to a Constant

If you have understood the above two types then this one is very easy to understand as its a mixture of the above two types of pointers. A constant pointer to constant is a pointer that can neither change the address its pointing to and nor it can change the value kept at that address.

```
const <type of pointer>* const <name of pointer>
const int* const ptr;
```

6. Applications of Volatile.

nxp semiconductor

There were 4 rounds

1.written test

(3 section)

1st section :problems on train, puzzles, time and distance , probability, ratios. And etc

2nd section : c programming

3rd section : related to os (10 questions).

2. Technical round 1

introduce yourself

1.can you assign any address to variable?

Ans:- no ,if it is a pointer variable then we will assign the address to the variable

2. What is volatile?

Ans:-

Volatile is key word used to prevent compiler to optimize a variable which can change unexpectedly beyound compilers comprehension.

3**. What is the difference between c and embedded c?**

C	embedded c
-- indirect access to register and memory	--- direct access to register and memory
-- exicututable file size is more	--- exicutable file size is less
(over head code size is more)	(over head code size is less)

4**.what is Inline function?**

Ans:--this function have a small definition and function body is substituted in each of the INLINE function.

5. What is code optimization?

Ans:-Optimization is a process of improving efficiency of a program in time (speed) or space (size).

6. Why is size of int is 4 bytes in linux and 2 bytes in Windows?

7**. Is there any other way you can optimize the code?**

3.technical round 2

1. What all the things you know about linux system programming?

2. When do you use mutex?

Ans:-Use for synchronization between the processes

mutex stands for mutual exclusion

3. When should you use static keyword?

Ans:-its used with variable as well as function

variable declared as static with in a file scope of that variable will be with in that loop.

Function decleared as static with in a module can be accessed by other function with in that module .

4. When should you use extern?

When you use *extern* keyword before the global variable declaration, the compiler understands you want to access a variable being defined in another program or file, and hence not to allocate any memory for this one. Instead, it simply points to the global variable defined in the other file. In this fashion, you can use the extern keyword in any number of programs to access the global variable *globalVar*.

Vvdn interview experience:

1st round:written test,total 70 questions

20questions from aptitude,30 questions from general electronics,20 questions from c all questions are in the fill in the blanks type.

2nd round:technical hr1

1)hardware architecture of linux os

2)all concepts in linux internals like ipc,semaphores,thread etc..
Ans:-

Interprocess Communication

Inter Process communication (IPC): Communication between processes is known as Inter Process communication. IPC is used for sending information from one process to another.

1 Shared Memory

One of the simplest interprocess communication methods is using shared memory. Shared memory allows two or more processes to access the same memory as if they all called malloc and were returned pointers to the same actual memory.When one process changes the memory, all the other processes see the modification.

2 Processes Semaphores

semaphores are counters that permit synchronizing multiple threads. Linux provides a distinct alternate implementation of semaphores that can be used for synchronizing processes (called process semaphores or sometimes System V semaphores). Process semaphores are allocated, used, and deallocated like shared memory segments.

3 Mapped Memory

Mapped memory can be used for interprocess communication or as an easy way to access the contents of a file. Mapped memory forms an association between a file and a process's memory.

4 Pipes

A pipe is a communication device that permits unidirectional communication. Data written to the "write end" of the pipe is read back from the "read end." Pipes are serial devices; the data is always read from the pipe in the same order it was written. Typically, a pipe is used to communicate between two threads in a single process or between parent and child processes.

4.1 FIFOs

A first-in, first-out (FIFO) file is a pipe that has a name in the filesystem.Any process can open or close the FIFO; the processes on either end of the pipe need not be related to each other. FIFOs are also called named pipes.

5 Sockets

A socket is a bidirectional communication device that can be used to communicate with another process on the same machine or with a process running on other machines.

Eg:-Internet programs such as Telnet, rlogin, FTP, talk, and the World Wide Web use sockets.

Mutexes

MUTual EXclusion locks.A mutex is a special lock that only one thread may lock at a time. If a thread locks a mutex and then a second thread also tries to lock the same mutex, the second thread is blocked, or put on hold. Only when the first thread unlocks the mutex is the second thread unblocked—allowed to resume execution.

Semaphore

A semaphore is a counter that can be used to synchronize multiple threads.As with a mutex, GNU/Linux guarantees that checking or modifying the value of a semaphore can be done safely, without creating a race condition.

Each semaphore has a counter value, which is a non-negative integer

A semaphore supports **two** basic operations:

1->A **wait operation** decrements the value of the semaphore by 1. If the value is already zero, the operation blocks until the value of the semaphore becomes positive (due to the action of some other thread).When the semaphore’s value becomes positive, it is decremented by 1 and the wait operation returns.

2->A **post operation** increments the value of the semaphore by 1. If the semaphore was previously zero and other threads are blocked in a wait operation on that semaphore, one of those threads is unblocked and its wait operation completes (which brings the semaphore’s value back to zero)

Processes Vs. Threads

->All threads in a program must run the same executable. A child process, on the other hand, may run a different executable by calling an exec function.

-> One process cannot corrupt another|->One thread can write the memory used
| by other thread

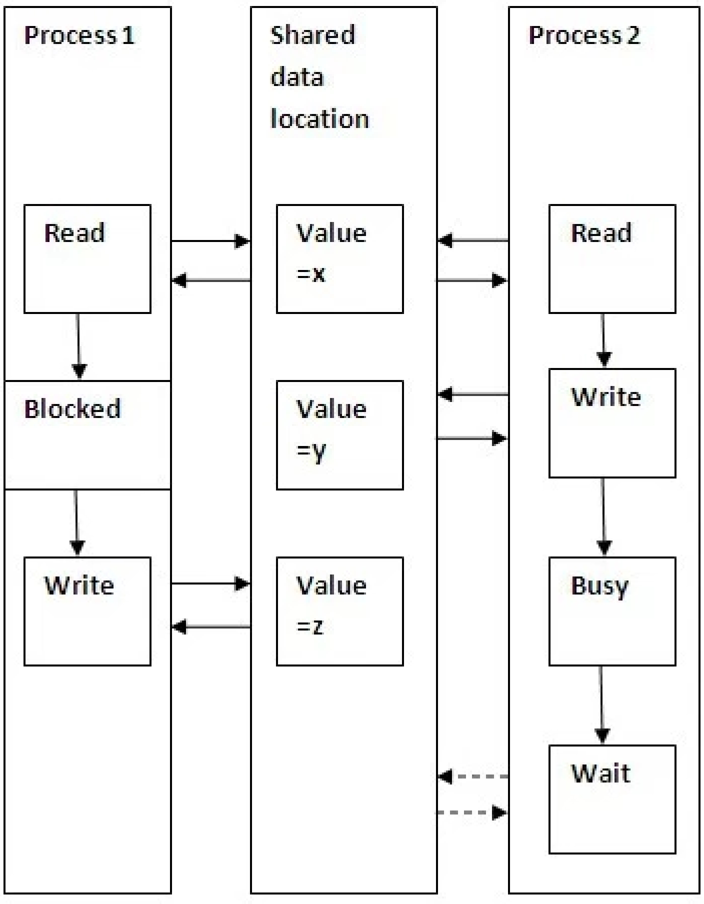
->Communication between processes |->Communication can be can be done using message passing done by sharing
| global variables.

->Don't share memory |->Threads within a process share memory

->Processes are not scheduled by OS |->Threads are scheduled by OS

->Process are passive. |->Threads are active.

Race condition: Race condition happens when two or more processes which are accessing same shared data and the output depends on the timing of particular processes. For understanding a race condition you have to consider two processes which are accessing the same shared data.



In the above given figure there are two processes which share the same data. Say, there is x data value which is stored in the shared location. Now, Process 1 has to finish the task of reading the data value and updating it. When Process 1 reads the location it gets a value x. So it stores the value x in its local variable. Now due to an interrupt Process 1 is blocked. On the other hand when Process 2 reads the value, it also gets x. So Process 2 also stores the value in their local variables. Then, Process 2 writes a value changing the value of x to y and saving the value in its local variable. Then, Process 2 starts doing other tasks. Meanwhile, Process 1 starts again from where it has left off. So it updates the data y which was written to the location with a new value of z. Now, the problem is that when Process 2 looks in the location it will never find the desired output.

3)7types of osi layers?
Ans:-

Physical (Layer 1)

- Data Link (Layer 2)
- Network (Layer 3)
- Transport (Layer 4)
- Session (Layer 5)
- Presentation (Layer 6)
- Application (Layer 7)

4)we need to explain all layers with protocal?

5)what is tcp and udp?
Ans:-

There are two types of Internet Protocol (IP) traffic. They are **TCP** or **Transmission Control Protocol** and **UDP** or **User Datagram Protocol**.

TCP is connection oriented – once a connection is established, data can be sent bidirectional.

UDP is a simpler, connectionless Internet protocol. Multiple messages are sent as packets in chunks using UDP.

TCP	**UDP**
->TCP is a connection-oriented protocol.	->UDP is a connectionless protocol.
->TCP is suited for applications that require high reliability, and transmission time is relatively less critical.	->UDP is suitable for applications that need fast efficient transmission, such as games.
->TCP rearranges data packets in the order specified.	->UDP has no inherent order as all packets are independent of each other. If ordering is required, it has to be managed by the application layer
->The speed for TCP is slower than UDP.	->UDP is faster because error recovery is not attempted. It is a "best effort" protocol.
->TCP header size is 20 bytes	->UDP Header size is 8 bytes.
->Acknowledgement segments	->No Acknowledgment

6)what is arp and rarp?

Ans:-

->**Address Resolution Protocol** is utilized for mapping IP network address to the hardware address that uses data link protocol.

- >**Reverse Address Resolution Protocol** is a protocol using which a physical machine in a LAN could request to find its IP address from ARP table or cache from a gateway server.

- >IP address of destination to physical address conversion is done by ARP, by broadcasting in LAN.

- >Physical address of source to IP address conversion is done by RARP.

- >ARP associates 32 bit IP address with 48 bit physical address.

- >Allowing a host to discover its internet address after knowing only its physical address is done by RARP.

7)what is the work of ICMP?

Ans:-

ICMP (Internet Control Message Protocol) is an error-reporting protocol network devices like routers use to generate error messages to the source IP address when network problems prevent delivery of IP packets. ICMP creates and sends messages to the source IP address indicating that a gateway to the Internet that a router, service or host cannot be reached for packet delivery. Any IP network device has the capability to send, receive or process ICMP messages.

8)write a socket program?

Ans:- claint

```
#include"header.h"
void main()
{
	char s[20];
	int len;
	struct sockaddr_in v,v1;
	int sfd,nsfd;
	sfd=socket(AF_INET,SOCK_STREAM,0);
	if(sfd<0)
	{
		perror("socket");
		return ;
	}
	perror("socket");
/////////////////////////////////////
	v.sin_family=AF_INET;
	v.sin_port=htons(2000);
	v.sin_addr.s_addr=inet_addr("127.0.0.1");
	len=sizeof(v);
	connect(sfd,(struct sockaddr *)&v,len);
	perror("connect");
/////////////////////////////////////
//	listen(sfd,5);
//	perror("listen");
//////////////////////////////////////
//	printf("before...\n");
//	nsfd=accept(sfd,(struct sockaddr *)&v1,&len);
//	perror("accept");
	printf("\nenter the data: ");
	scanf("%s",s);
	write(sfd,s,strlen(s)+1);
}
```

output:----------------------------

Ans:-server

```
#include"header.h"
```

```
void main()
{
        char s[20];
        int len;
        struct sockaddr_in v,v1;
        int sfd,nsfd;
        sfd=socket(AF_INET,SOCK_STREAM,0);
        if(sfd<0)
        {
                perror("socket");
                return ;
        }
        perror("socket");
////////////////////////////////////
        v.sin_family=AF_INET;
        v.sin_port=htons(2000);
        v.sin_addr.s_addr=inet_addr("0.0.0.0");
        len=sizeof(v);
        bind(sfd,(struct sockaddr *)&v,len);
        perror("bind");
////////////////////////////////////
        listen(sfd,5);
    perror("listen");
/////////////////////////////////////
        printf("before...\n");
        nsfd=accept(sfd,(struct sockaddr *)&v1,&len);
        perror("accept");
        read(nsfd,s,sizeof(s));
        printf("data:%s\n",s);
}
```

9)how to assign ip addresses based on different classes?
10)they gave a ip address we need to find out that ip address belongs to which class using c program?

Ans:-

```
#include<stdio.h>
/*      CLASSES
A:        0 - 127 ( = 128 )
B:      128 - 191 ( =  64 )
C:      192 - 223 ( =  32 )
D:      224 - 239 ( =  16 )
E:      240 - 255 ( =  16 )
 */
int main()
{
        int a[4],i=0;
        printf("Enter The IP address xxx.xxx.xxx.xxx :-");
        for(i=0;i<4;i++)
                scanf("%d",&a[i]);
        printf("\n IP ADDRESS:%d.%d.%d.%d",a[0],a[1],a[2],a[3]);
        printf("The IP address is in Class: ");
        if(a[0]>=0 && a[0]<=127)
                printf("Class A");
        if(a[0]>127 && a[0]<191)
                printf("Class B");
        if(a[0]>191 && a[0]<224)
                printf("Class C");
        if(a[0]>224 && a[0]<=239)
                printf("Class D");
        if(a[0]>239)
                printf("Class E");
        return 1;
}
```

11)designing,coding,testing for ATM machine?

3rd round: tecnical Hr2

1)divide 2 numbers without using divison operator?

Ans:-

```
#include<stdio.h>
int main()
{
        int num1, num2, a, b, count = 0 ,ret;
        printf("\nEnter First Number:\t");
        scanf("%d", &a);
        printf("\nEnter Second Number:\t");
        scanf("%d", &b);
        num1 = a;
        num2 = b;
        while(num1 >= num2)
        {
                ret  = num1 - num2;
                count++;
                num1 = num1 - num2;
        }
        printf("\nDivision of %d and %d:\nQuotient:\t%d\nRemainder:\t%d\n", a, b,
count,ret);
        return 0;
}
```

2)same question for multiplication without using * operator?

Ans:-

```
#include<stdio.h>

int main()
{
	int num1, num2, a, b, count = 0 ,ret;
	printf("\nEnter First Number:\t");
	scanf("%d", &a);
	printf("\nEnter Second Number:\t");
	scanf("%d", &b);
	num1 = a;
	num2 = b;
	for(int i=num2;i>0;i--)
	{
		ret  = ret + num1 ;
	}
	printf("\nDivision of %d and %d:\tmul:%d\n", a, b,ret);
	return 0;
}
```

3)**they gave an ip address 192.168.1.1and 10.10.10.10 using atoi print that same ip address with dots properly?**

4)from digital electrons all basics like boolien expession solution,k map,logic gates,sequential and combinational differencd,mux and demux,encoder and decoder,flipflops T,SR,JK with truth tables

Ans:-

Multiplex	-> Many to One
DeMultiplex	-> One to Many
Encode	-> Converting a message to number
Decode	-> Converting a number to message.

In digital Electronics Multiplexer convert many signals through one wire. There are multiple types multiplexing are there like Time Domain Multiplexing, Frequency Domain Multiplexing

5)difference between microprocessor and controller?

<u>Microprocessor</u> **<u>Micro Controller</u>**

->Microprocessor is heart of Computer system.| ->Micro Controller is a heart of embedded system.

->It is just a processor. Memory and I/O | ->Micro controller has external processor along

components have to be connected externally | with internal memory and i/O components
->the circuit becomes large. | ->the circuit is small.
->Cannot be used in compact systems and | ->Can be used in compact systems and hence it is
hence inefficient | an efficient technique
->Cost of the entire system increases | ->Cost of the entire system is low
->Due to external components, the entire | ->external components are low, total power
power consumption is high. | consumption is less.
->Microprocessor have less no.of registers, | ->Micro controller have more number of registers
hence more operations are memory based. | hence the programs are easier to write.
->Mainly used in personal computers. | ->Used mainly in washing machine, MP3 players
____

6)explain architecture of 8085 and 8051?circuts we need to solve?
7)from google they will ask some puzzles for checking our confidence

4th round: HR

1)tell me about your self?
2)how many years will you work for this company?
3)what is +ves and -ves?
4)what do you know about VVDN?

<u>Valeo India interview experience</u>

1.written test: please see the attachment also...
1.Swap all even and odd bits of a number
2.implement user defined sizeof
Ans:-
```
#define my_sizeof(type) (char *)(&type+1)-(char*)(&type)
int main()
```

```
{
        double x;
        printf("%d", my_sizeof(x));
        getchar();
        return 0;
}
```

3.Delete the node without using hptr.(discussed by Tandava sir)

4.Oneline c function to find the given number is power of 2 or not

Ans:-

```
#include<stdio.h>
#define bool int

/* Function to check if x is power of 2*/
bool isPowerOfTwo (int x)
{
/* First x in the below expression is for the case when x is 0 */
        return x && (!(x&(x-1)));
}

/*Driver program to test above function*/
int main()
{
        isPowerOfTwo(31)? printf("Yes\n"): printf("No\n");
        isPowerOfTwo(17)? printf("Yes\n"): printf("No\n");
        isPowerOfTwo(16)? printf("Yes\n"): printf("No\n");
        isPowerOfTwo(2)? printf("Yes\n"): printf("No\n");
        isPowerOfTwo(18)? printf("Yes\n"): printf("No\n");
        isPowerOfTwo(1)? printf("Yes\n"): printf("No\n");

        return 0;
}
```

5.pascal triangle pattern

6.Print the word of the number without using any if-else and switch conditions
Example :45 = Four Five

7.convert integer into string

8.write a c program to count number of zeros in the given number

9.program to find the middle node of a linked list(optimized code explained by Tandava sir)

10.find the sum of the diagonal elements in matrix without transversing whole matrix.

2.technical round:
->project block diagram n explaination(alrdy discussed by Badri sir)
->i2c/spi/uart
->calloc=malloc+(find here wat to write)
->strcpy userdefined to be worked for all cases i.e generic inputs n all boundary conditions(i didnt get this well)
->interrupt lyf cycle(interrupt latecncy and respone time as well)
->duty cycle
->write an embedded program to generate 5ms delay using T0M0(timer 0 mode 0)

written test....................
10 programs....
1.swap two arrays and reverse them.
Logic:-

```
printf("swap two arrays.....\n");
	for(i=0;i<num;i++)
	{
		temp=arrayp1[i];
		arrayp1[i]=arrayp2[i];
		arrayp2[i]=temp;
	}
 printf("reverse....\n");
	for(i=num-1,j=0;i>j;i--,j++)
	{
		temp=arrayp1[i];
		arrayp1[i]=arrayp1[j];
		arrayp1[j]=temp;

		temp=ap2[i];
		arrayp2[i]=arrayp2[j];
		arrayp2[j]=temp;
	}
```

2.reverse print the linked list.

3.print a pattern

```
#include<stdio.h>

int main()
{
	int i,j,k,n;
	int num;
```

```
        printf("\nenter the number of rows: ");
        scanf("%d",&num);

        for(i=0;i<=num-1;i++)
        {
                for(j=num-1;j>i;j--)
                        printf(" ");

                for(k=i;k>=0;k--)
                        printf("*");

                for(n=i-1;n>=0;n--)
                        printf("*");

                printf("\n");
        }
 for(i=0;i<=num-1;i++)
        {
                for(n=i;n>=0;n--)
                        printf(" ");

                for(j=num-2;j>i;j--)
                        printf("*");

                for(j=num-2;j>=i;j--)
                        printf("*");

                printf("\n");
        }


}
```

output:

```
enter the number of rows: 6
     *
    ***
   *****
  *******
 *********
***********
 *********
  *******
   *****
    ***
     *
```

4.convert decimal to bcd

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
	int i,res,dec_num,binary_num;
	printf("enter the decimal number .....\n");
	scanf("%d",&dec_num);
	  for(i=31;i>=0;i--)
	  {
		res=dec_num>>i&1;
		printf("%d",res);
	  }
}
```

5.swap two numbers without using arithmetic operators

6.print the endianness and make it little endian to big endian and vice versa

7.design a calculator(using unsigned char) with basic operations

8.write a function to put the binary representation of an unsigned char in an array and return the array.
9.there is an array of 101 numners which consists of numbers from 1 to 100 with the number repeated.Find the repeated number in minimum iterations.
10.delete all occurences of given element from the array and move tha array upwards.
eg. after deleting 3 from 2,3,3,4,3 the output array should be 2,4,0,0,0.
50 objective questions both c and 8051 based qs(vector internals type) plus logical reasoning 10 qs.

technical interview...............
1.tell me about urself
2.tell me about ur family background
3.what is the diff between general os and real time os
4.explain interrupt cycle
5.difference between interrupt latency and interrupt response time,explain using diagram
6.draw connections for spi and uart
7.explain the diagram drawn
8.what is difference between asynchronous and synchronous
9.userdefined strcpy,strcat,calloc
take care of errors and boundary conditions
10.explain real time os
11.what is priority inversion
12.what is the diff between process and thread

13.how can we do process synchronization
14.what is difference between semaphore and mutex
15.when 50% of isr is executed a non maskable interrupt of same priority comes up.how do u manage the situation.
16.int *ptr1=1000,*ptr2=2000;

```
    int *ptr3=ptr1+ptr2;
```

find the output...........This is wat am able to collect.
17.can u stay in chennai.any of ur relatives stay here?
18.what kind of work are u expecting here?
19.Any qs??

HR round...

It is a formality round.A HR test will be conducted in which u will be asked 30

<u>Valeo [Interview experience]</u>

1.Written Test

In written there was 40 questions - few objective type
and
few Q&A type
Most of the questions(75%) were asked on testing.
Remaining (25%) on c language for finding output and errors.

2.Technical Round

Technical round was done for 1hr:15min
Questions were based on **<u>Testing</u>**: For testing refer these topics:
a.Types of testing
b.Software Development Life cycle.
c.Testing life cycle.
d.Test cases.
e.Black box testing,White box testing,Gray box testing.
f.statement coverage,Decision coverage,path coverage.
g.learn to write flow diagrams and state diagrams for a given test condition.
Note: No need to worry about testing because there will be having classes on testing also in vector for 3 days for those who are appearing in testing positions.

<u>Puzzles</u>: a.puzzles were asked like finding a 3 digit password by giving some conditions
b.asked to write a series of numbers by giving an example condition.
They asked me nearly 10 puzzle questions and are easy if we carefully observe it

C-Programming: 1.Asked me to write logic for counting number of repeating patterns of 0s
1s in a given binary number.
ex: In "110101010110010101110" count number of times 010 occurs.
2.Write program to display binary,hexa,octal of a given integer number.
3.Linked list program to convert binary to decimal.
4.Small logics like set,clear,compliment bits.

3.Hr Round

In HR they asked
about myself and family,What i know about

Valeo
,Why did i attended valeo. It was very formal and mostly they
wanted learning and working attitude in the candidate and to know how much committed we are for the company.

Aricent Written / Interview Experience

I, D Sujitha (V16HE4D8) attended for Aricent Written and Interview. The result is not yet announced.

Following some of the topics / questions covered in the written / interview

1. Written Test (120 min)
a. Predict the output in C Programming
b. Programming on Linked List, Binary Tree, Hash Table
c. Algorithamic based programming (Arrow / Target -- Dot Board Game)
d. Print the number of occurrences of a number in array.
e. CPU Utilization / Average Time / Turnaround Time / CPU Scheduling from Linux
f. Fragmentation

2. Interview
a. C Basics including Strings / User Defined Functions
b. Fork / vfork difference
c. mutex and semaphore difference
d. deadlock / threads
e. Write an algorithm to find the 5th element from last in a linked list
f. Difference between union and structure / Structure Padding / Sizeof Structure
g. use of pointers
h. C++ Basics / Class Example
i. Inheritance with Example
j. Access specifiers with example
k. How to know the given compiler is 32bit or 64bit without seeing the system

properties
l. About All Projects mentioned in CV along with the Block Diagrams

And this questions are very important

```
“static int a=10;
int main()
{
int a=10;
printf(“%d\n”,a);
//I want print global variable value how to u print that value
}
o/p :-
----------------------------
int main()
{
static int a=10;
static int b=a;
printf(“%d\n”,b);
}
---------------------
int main()
{
static int a=10;
static int b=&a; // error  o/p
printf(“%d\n”,b);
..............................
int c=10;
static int *p=&c;
.............................
static int c=10;
static int *p=&c;
}
```

```
int main()
{
printf("%d\n");
}
```

//Please find the answer friends and reason also because three interview i faced this program

and concentrate more on storage class this will help you lot friends"

4.***Technical round 3(in this process only he discussed two and off hour)***

1.what is tcp/ip

2.what are the functions involved in the socket programming.

2.Can you write socket program

3.Difference b/w tcp/udp

4. What all the things you know about Linux system programming?

*5.Explain the virtual memory

6.What are the IPC mechanism are there? Explain in detail

7.without using semaphore how to synchronize the process.

*8.When do you use mutex?

*9.what is mean by race condition.(in this topic he discussed half hour)

*10.what is mean by interrupt latency

11.what is mean by RTOS

*12.Difference b/w RTOS and OS

13.what is mean by friend function where it's used.using friend function write any program.

15.what is mean by constructor and destructor
*14. When should you use static keyword?
*15.When should you use extern?

16.Booting steps of OS? And some question s regarding Run level ?

*16.I got executable in intel processor if im executed in the ARM processor means what will happen.

*{They are concentrating more on c language only so please prepare c as much as possible friends.Not only this company all company's focus on C language only)

Written test consists of two sections

1) **Aptitude** which consists of 10 questions and the level of difficulty is easy and no –ve marking.

Question follows as:

1) Waste material shipped is 6.25% that is 450. Then total shipped material is....?

2) a-b/3.5=4/5 which is greater a or b?

3) Radius of the circle is increased by 6% then total area increased is...?

4) Area of the right angle triangle has to find out for a given side...?

5) 3t=4s then 5t/6s=13/4. Then what is s in terms of t...?

And questions from number system, probability etc...

2nd section is the combination of c and embedded c

```
struct tag{
 int i=5;
 char c=a;
flaot f=12.5;
};
```

ans: eror

when reset will be activated?

for which instruction stack is affected

a) call & ret b) cal & ref c) push and pop d) none

One question based on command line argument;

3-4 questions on loops

```
printf("%d",-2>>2)
```

2 questions on strings arrays?

Two questions on pointers.

```
int i=1.2
 ceil(i)=....? Float (i)=....?
ans:  2,1
```

Some questions on embedded C

They announced results after two days

2nd round: technical

There are three persons in the cabin, firstly they asked are you relocate, and then after they started technical questions based on C, Embedded C, communication, tcp/ip, Cpp.

1. What is printf?
2. How to write in to character array?
3. What is fprintf? What is use of it?
4. what is simplex, half duplex, full duplex?
5. what is structure?
6. how can u differentiate structure and union ,array/in printf there is a only string (printf("vector India")) not %d or %c then which function is used by compiler that is printf or something.

   Ans **putc**
7. what is linked list? how it is different from array apart from memory allocation?
8. how can u delete a randomly given node ,without head node?
9. what is sorting? types of sorting?
10. which is the worst sorting technique?
11. what is time complex city for sorting techniques?
12. what is inline function?
13. what is memory leak?

14. what is structure padding? how to remove it?and some of the basic programs.

15. what is malloc? what it returns?

16. like realloc?

17. what printf returns?

8051 questions

1. what is interrupt?
2. what is polling?
3. difference between them?
4. explain about can protocol?
5. **explain I2C protocol?**

Inter-Integrated Circuit

The I 2 C-bus (“eye-squared-see bus”), is a simple, two-wire serial interface that provides the communications link between integrated circuits in a system

The data wire (SDA) carries data, while the clock wire (SCL) synchronizes data transfers.

Master-slave hierarchy

I2C-bus devices are classified as master or slave. Masters initiate a message and slaves respond to a message.

Multiple devices

The I 2 C-bus is designed to support multiple devices. Each I 2 C-bus slave device has a unique slave address. When a master sends a message, it includes the slave address at the beginning of the message. All devices on the bus hear the message, but only the addressed slave responds to it.

I2C start condition

• Master pulls SDA low while SCL is high

• Normal SDA changes only happen when SCL is low

I2C stop condition

• Master pulls SDA high while SCL is high

• Also used to abort transactions

6. **explain about SPI protocol?**

Serial Peripheral Interface

Serial bus protocol ,Fast, easy to use, and simple , Very widely used, Not "standardized"

A 4-wire communications bus

• Typically communicate across short distances

• Supports

– Single master

– Multiple slaves

• Synchronized

– Communications are "clocked"

SPI Capabilities

• Always full-duplex

– Communicates in both directions simultaneously

– Transmitted (or received) data may not be meaningful

• Multiple Mbps transmission speeds

– 0-50 MHz clock speeds not uncommon

• Transfer data in 4 to 16 bit characters

• Supports multiple slaves

SPI bus wiring

Bus wires

Master-Out, Slave-In (MOSI)

Master-In, Slave-Out (MISO)

System Clock (SCLK)

Slave Select/Chip Select (SS1#, ..., SS#n or CS1, ..., CSn)

• Master asserts slave/chip select line

• Master generates clock signal

• Shift registers shift data in and out

SPI signal functions

• MOSI – carries data out of master to slave

• MISO – carries data out of slave to master

– Both MOSI and MISO are active during every transmission

• SS# (or CS) – unique line to select each slave chip

• SCLK – produced by master to synchronize transfers

7. explain about CAN BUS protocol?

The Controller Area Network (CAN) is a serial communications protocol which efficiently supports distributed realtime control with a very high level of security.

CAN Standards

CAN 2.0A

Standard CAN (ISO 11898)

11-bit Identifier

1 Mbps

CAN 2.0B

Extended CAN (ISO 11519)

29-bit Identifier

125 kbps

CANbus Node

Each node requires a:

- Central processing unit, microprocessor, or host processor
  - The host processor decides what the received messages mean and what messages it wants to transmit.
  - Sensors, actuators and control devices can be connected to the host processor.
- CAN controller; often an integral part of the microcontroller
  - Receiving: the CAN controller stores the received serial bits from the bus until an entire message is available, which can then be fetched by the host processor (usually by the CAN controller triggering an interrupt).
  - Sending: the host processor sends the transmit message(s) to a CAN controller, which transmits the bits serially onto the bus when the bus is free.
- Transceiver Defined by ISO 11898-2/3 Medium Access Unit [MAU] standards
  - Receiving: it converts the data stream from CANbus levels to levels that the CAN controller uses. It usually has

CAN has the following properties

• prioritization of messages

• guarantee of latency times

• configuration flexibility

• multicast reception with time synchronization

• system wide data consistency

• multimaster

• error detection and signalling

• automatic retransmission of corrupted messages as soon as the bus is idle again

• distinction between temporary errors and permanent failures of nodes and autonomous switching off of defect nodes

Priorities

The IDENTIFIER defines a static message priority during bus access.

Multimaster

When the bus is free any unit may start to transmit a message. The unit with the message of higher priority to be transmitted gains bus access.

Arbitration

Whenever the bus is free, any unit may start to transmit a message. If 2 or more units start transmitting messages at the same time, the bus access conflict is resolved by bitwise arbitration using the IDENTIFIER. The mechanism of arbitration guarantees that neither information nor time is lost.

If a DATA FRAME and a REMOTE FRAME with the same IDENTIFIER are initiated at the same time, the DATA FRAME prevails over the REMOTE FRAME. During arbitration every transmitter compares the level of the bit transmitted with the level that is monitored on the bus. If these levels are equal the unit may continue to send. When a ’recessive’ level is sent and a ’dominant’ level is monitored (see Bus Values), the unit has lost arbitration and must withdraw without sending one more bit.

Frame Types

Message transfer is manifested and controlled by four different frame types:

A DATA FRAME carries data from a transmitter to the receivers.

A REMOTE FRAME is transmitted by a bus unit to request the transmission of the

DATA FRAME with the same IDENTIFIER.

An ERROR FRAME is transmitted by any unit on detecting a bus error.

An OVERLOAD FRAME is used to provide for an extra delay between the preceding

and the succeeding DATA or REMOTE FRAMEs.

DATA FRAMEs and REMOTE FRAMEs are separated from preceding frames by an INTERFRAME SPACE.

IDENTIFIER

The IDENTIFIER’s length is 11 bits. These bits are transmitted in the order from ID-10 to ID-0. The least significant bit is ID-0. The 7 most significant bits (ID-10 - ID-4) must not be all ’recessive’.

Difference Between CAN, LIN, I2C

protocol	I2C	SPI	CAN
Protocol	Synchronous interface used on PCB	Synchronous interface used on PCB	CAN is Asynchronous Interface & uses wires for long distance communications.
Invented	Philips	Motorola	Bosch
Data rate	I2C Supports Speed is : 100Kbps(Standard) :400Kbps(Fast) :3.4Mbps(High Speed)	SPI Supports : 3Mbps to 10Mbps	10Kbps to 1 Mbps
master	I2C is multi-master, Address based Communication	SPI is Master Slave, With Slave select(SS) based Communication	MultiMate , Message based communications, Reliable
Pins required	I2C needs 2 pins	SPI needs 3+n pins (n is no. of devices)	CAN H and CAN L
supports	I2C supports 127 devices	limited by available Chip selects	2043 devices
communication	I2C is half duplex as there are only two lines(SCL and SDA)	SPI is Full Duplex as between a Master and a dedicated slave as selected by slave select signal;	Half duplex
	whereas the I2c is generally used in same PCB	whereas the SPI is generally used in same PCB	CAN is generally used for device net (different device at different location)
	bus arbitration is possible in case of i2c.	not in case of spi.	

noise sensitivity of i2c is high... there is chance to corrupt the r/w bit... then whole data is corrupted		But in case of spi.. Chance is very less as whole word is transmitted.
Single duplex	Full duplex operation	Half duplex

8. Explain about your project?
9. frame structure of uart?
10. **what is volatile?**

volatile

The modifier volatile tells the compiler that a variable's value may be changed in ways not explicitly specified by the program.

For example, a global variable's address may be passed to the operating system's clock routine and used to hold the system time. In this situation, the contents of the variable are altered without any explicit assignment statements in the program.

This is important because most C compilers automatically optimize certain expressions by assuming that a variable's content is unchanging if it does not occur on the left side of an assignment statement; thus, it might not be reexamined each time it is referenced. Also, some compilers change the order of evaluation of an expression during the compilation process. The volatile modifier prevents these changes.

You can use const and volatile together.

const volatile char *port = (const volatile char *) 0x30;

11. **what does compiler do if volatile keyword is specified? with example apart from a++ ++a ?**

The declaration of a variable as volatile tells the compiler that the variable can be modified at any time externally to the implementation

12. use of unions practical example?
13. what is interrupt latency?

14. what is subroutine?

Difference between macro and subroutine?

- Subroutine: 1.When a task is to be done repeatedly then it is written assubroutine and this subroutine will be called each time to perform that task.

- Subroutine program will be stored in some memory location and program control will be transfered to that location each time.

- where as in macro the number of instructions will be less than subroutine.Here each time u call a macro that set of instructions will be inserted in that location.

- macro doesn't have return statement while subroutine has.

- memory requirement for macro is higher.

- execution time of macro is lesser than subroutine.

15. What is Memory Leak? How can we avoid?

Memory leak occurs when programmers create a memory in heap and forget to delete it.
Memory leaks are particularly serious issues for programs like daemons and servers which by definition never terminate.

some of the questions asked to my friend are

1.draw the block diagram of 8051? and explain?

2.difference between C and Embedded c?

you should be in a position to answer each and every question from your resume? o/w it was a bad impression?

4.difference between polling and interrupt?

some of basic programs like set clear complement bit?

which operator u use for setting a bit and how?

what represents dominant mode and ressive mode.

how can u differentiate standard frame and extended frame in the transmitter or rx stage?

3rd round: HR

This was a formal round in which he explained about company norms, terms and conditions. Discussed about bond which was about 3 years. 1.5L bond breakage amount as bank guaranty Timings of the company and asked any questions.

---

4. What is pre-emptive and non-pre-emptive scheduling algorithms?

Non pre-emptive scheduling: When the currently executing process gives up the CPU voluntarily.

Pre-emptive scheduling: When the operating system decides to favor another process (High priority), pre-empting the currently executing process.

Briefly explain the following algorithms:

FCFS:

1. Round Robin – Time slice based

In RR time slices are assigned to each process in equal portions and in circular order, handling all processes without priority (also known as cyclic executive). Round-robin scheduling is simple, easy to implement, and starvation-free.

2. Round Robin – Priority based

It works same as RR time slice, but we can also assign priority to process. More priority means large time slice.

Priority based:

1. Rate monotonic

Rate-monotonic scheduling (RMS) is a scheduling algorithm used in real-time operating

systems (RTOS) with a static-priority scheduling class. The static priorities are assigned according to the cycle duration of the job, so a shorter cycle duration results in a higher job priority. Smaller the period, higher the priority.

2. Earliest deadline first

Earliest deadline first (EDF) or least time to go is a dynamic scheduling algorithm used in real-time operating systems. A priority queue will be maintained and will be searched for the process closest to its deadline. This process is the next to be scheduled for execution.

Q.What is a real-time systems (RTS)? What are the different types of RTS?

Real Time Operating System (RTOS) is designed to provide a predictable (normally described as deterministic) execution pattern. This is particularly of interest to embedded systems as embedded systems often have real time requirements. A real time requirement is one that specifies that the embedded system must respond to a certain event within a strictly defined time (the deadline). A guarantee to meet real time requirements can only be made if the behavior of the operating system's scheduler can be predicted.

<u>Soft-RTOS</u>

Soft real time systems can miss some deadlines, but eventually performance will degrade if too many are missed. Example – multimedia streaming, computer games

<u>Hard-RTOS</u>

Hard real-time means you must absolutely hit every deadline. Very few systems have this requirement. Some examples are nuclear systems, some medical applications such as pacemakers, a large number of defense applications, avionics etc. Where life critical situation we have to use hard real time system.

7. Explain differences between General Purpose Operating System (GPOS), Real Time Operating System (RTOS) and Embedded Operating System (EOS)

<u> GPOS</u>

General Purpose Operating System (GPOS) is a software that provides interface to the user to access various applications. These are running in a commonly used systems & run applications like desktop PCs (ex: Windows 10). They run on a general purpose hardware.

<u> RTOS</u>

Real Time Operating System (RTOS) is a software that has certain time constraint. Such OS operate with stringent time expectations to deliver a predictable response to use applications(ex: robotics). This OS can run either on a general or specific hardware that is designed.

<u>->EOS</u>

Embedded OS (EOS) is a software that runs on a specific hardware that runs specific set of applications. Along with running in a specific target, EOS will have resource constraints (ex: size) hence it should be customizable and configurable to meet embedded system needs.