

Written Test / Interview Preparation Guide

Interview / Written test overview

First Round Details

Second Round Details

HR interview details

Preparation Materials



Emertxe Information Technologies - Placement Department - Preparation Purpose Only

1. What are threads? How are they different from processes?

A thread is the smallest unit of processing that can be performed in an OS. In most modern operating systems, a thread exists within a process - that is, a single process may contain multiple threads.

	Process	Thread
Definition	An executing instance of a program is called a process.	A thread is a subset of the process.
Process	It has its own copy of the data segment of the parent process.	It has direct access to the data segment of its process.
Communication	Processes must use inter-process communication to communicate with sibling processes.	Threads can directly communicate with other threads of its process.
Overheads	Processes have considerable overhead.	Threads have almost no overhead.
Creation	New processes require duplication of the parent process.	New threads are easily created.
Control	Processes can only exercise control over child processes.	Threads can exercise considerable control over threads of the same process.
Changes	Any change in the parent process does not affect child processes.	Any change in the main thread may affect the behavior of the other threads of the process.
Memory	Run in separate memory spaces.	Run in shared memory spaces.
File descriptors	Most file descriptors are not shared.	It shares file descriptors.
File system	There is no sharing of file system context.	It shares file system context.
Signal	It does not share signal handling.	It shares signal handling.
Controlled by	Process is controlled by the operating system.	Threads are controlled by programmer in a program.

Dependence	Processes are independent.	Threads are dependent.
-------------------	----------------------------	------------------------

2. What are the different IPC mechanisms?

- Pipes (half duplex)
- FIFOs (named pipes)
- Stream pipes (full duplex)
- Named stream pipes
- Message queues
- Semaphores
- Shared Memory
- Sockets
- Streams

3. What is mutex and what is a semaphore? Also tell the difference between them.

BASIS FOR COMPARISON	SEMAPHORE	MUTEX
Basic	Semaphore is a signaling mechanism.	Mutex is a locking mechanism.
Existence	Semaphore is an integer variable.	Mutex is an object.
Function	Semaphore allow multiple program threads to access a finite instance of resources.	Mutex allow multiple program thread to access a single resource but not simultaneously.
Ownership	Semaphore value can be changed by any process acquiring or releasing the resource.	Mutex object lock is released only by the process that has acquired the lock on it.
Categorize	Semaphore can be categorized into counting semaphore and binary semaphore.	Mutex is not categorized further.
Operation	Semaphore value is modified using wait() and signal() operation.	Mutex object is locked or unlocked by the process requesting or releasing the resource.

Resources Occupied	If all resources are being used, the process requesting for resource performs wait() operation and block itself till semaphore count become greater than one.	If a mutex object is already locked, the process requesting for resources waits and queued by the system till lock is released.
---------------------------	---	---

4. Why do we need synchronization?

An operating system needs synchronization because of concurrency, interrupts, and preemption.

If a computation can be interrupted while a shared variable is being accessed, and the interrupting computation can interact with that shared variable, then the results of the original computation can seemingly produce non-sensual results. To prevent this, we protect all accesses to shared variables with synchronization primitives, creating critical sections that cannot be concurrently accessed, thereby ensuring the integrity of all computations.

5. How do you create child processes in linux?

The process which creates a new process called parent process. To create a child process we can use fork() system call. Fork will create a child process which is a duplicate of parent process with a new process id.

```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<sys/wait.h>
int main()
{
    int status;
    int pid;
    pid=fork();
    if(pid<0)
    {
        printf("\n Error ");
        exit(1);
    }
    else if(pid==0)
    {
        printf("\n Hello I am the child process ");
        printf("\n My pid is %d ",getpid());
        exit(0);
    }
    else
    {
        wait(&status);
        printf("\n Hello I am the parent process ");
    }
}
```

```
printf("\n My actual pid is %d \n ",getpid());  
exit(1);  
}  
}
```

6. What are the various system calls that you know?

There are 5 different categories of system calls. They are process control, file manipulation, device manipulation, information maintenance and communication.

List of system calls: Refer link: <http://man7.org/linux/man-pages/man2/syscalls.2.html>

7. Explain makefiles? And what will you write inside a makefile?

Compiling the source code files can be tiring, especially when you have to include several source files and type the compiling command every time you need to compile. Makefiles are the solution to simplify this task. Makefiles are special format files that help build and manage the projects automatically. Makefiles are a simple way to organize code compilation.

A makefile is useful because (if properly defined) allows recompiling only what is needed when you make a change. In a large project rebuilding the program can take some serious time because there will be many files to be compiled and linked and there will be documentation, tests, examples etc. When you work on the project and you make a little change it would be annoying having to wait to rebuild everything each time.

A makefile store a list "input" files, "output" files and "commands" needed to produce the output given the input. When you make a change to the project the command make will check the date of the input files with the date of the corresponding output files and if the input files has been changed it will recreate the corresponding output by running the command.

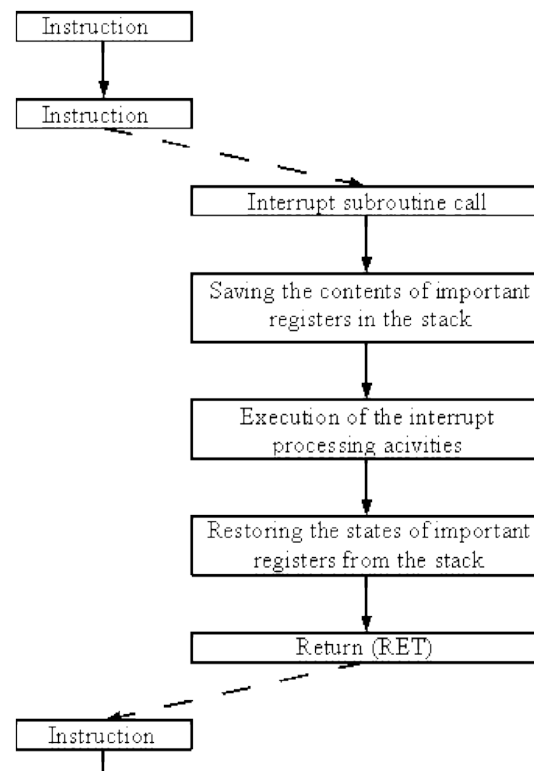
What will you write inside a makefile- refer link below:

<http://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/>

8. What happens when an interrupt occurs?

Answer 1:

1. Any currently executing instruction is completed.
2. The PC, which points to the next instruction, is pushed onto the stack.
3. The SR is pushed onto the stack.
4. The interrupt with highest priority is selected if multiple interrupts occurred during the last instruction and are pending for service.
5. The interrupt request flag resets automatically on single source flags. Multiple source flags remain set for servicing by software.
6. The SR is cleared. This terminates any lower power mode. Because the GIE bit is cleared, further interrupts are disabled.
7. The content of the interrupt vector is loaded into the PC. The program continues with the interrupt service routine at that address.



Answer 2:

When an interrupt is triggered, the following actions are taken automatically by the microcontroller:

1. The current Program Counter is saved on the stack, low-byte first.
2. Interrupts of the same and lower priority are blocked.
3. In the case of Timer and External interrupts, the corresponding interrupt flag is cleared.
4. Program execution transfers to the corresponding interrupt handler vector address.
5. The Interrupt Handler Routine executes.

If the interrupt being handled is a Timer or External interrupt, the microcontroller automatically clears the interrupt flag before passing control to your interrupt handler routine. This means it is not necessary that you clear the bit in your code.

9. What are the various types of interrupts?

Interrupts we can be divided into two categories.

Hardware interrupts and software interrupts.

Interrupts generated due to hardware changes (ex: USB plug-in to your PC) is called hardware interrupt. These are used to handle asynchronous hardware changes or data

manipulation. An interrupt generated by software/instruction is called software interrupt. Eg: INT 0x80, sys_enter. These we are using to implement system calls. All the system calls you study as a part of Linux Internals course is also soft interrupts. In both case execution will jump to ISR.

Another categorization of interrupts are mask-able and unmask-able interrupts.

Assume when you executing inside an ISR and another interrupts comes. Default case it will stop the current ISR and start executing new ISR. This is a problem if you doing an important job (ex: critical real time task like air bag control) in ISR. So there is an option to change this by masking interrupts. But it's not possible for all interrupts available. So the interrupts which is possible to mask or block is called mask-able interrupts, and which is not possible to mask is non mask-able interrupts.

10. What is stored in interrupt vector table?

An "interrupt vector table" (IVT) is data structure that associates a list of interrupt handlers with a list of interrupt requests in a table of interrupt vectors. Each entry of the interrupt vector table, called an interrupt vector, is the address of an interrupt handler.

Table 6-1. Exceptions and Interrupts

Vector No.	Mnemonic	Description	Source
0	#DE	Divide Error	DIV and IDIV instructions.
1	#DB	Debug	Any code or data reference.
2		NMI Interrupt	Non-maskable external interrupt.
3	#BP	Breakpoint	INT 3 instruction.
4	#OF	Overflow	INTO instruction.
5	#BR	BOUND Range Exceeded	BOUND instruction.
6	#UD	Invalid Opcode (UnDefined Opcode)	UD2 instruction or reserved opcode. ¹
7	#NM	Device Not Available (No Math Coprocessor)	Floating-point or WAIT/FWAIT instruction.
8	#DF	Double Fault	Any instruction that can generate an exception, an NMI, or an INTR.
9	#MF	CoProcessor Segment Overrun (reserved)	Floating-point instruction. ²
10	#TS	Invalid TSS	Task switch or TSS access.
11	#NP	Segment Not Present	Loading segment registers or accessing system segments.
12	#SS	Stack Segment Fault	Stack operations and SS register loads.
13	#GP	General Protection	Any memory reference and other protection checks.
14	#PF	Page Fault	Any memory reference.
15		Reserved	
16	#MF	Floating-Point Error (Math Fault)	Floating-point or WAIT/FWAIT instruction.
17	#AC	Alignment Check	Any data reference in memory. ³
18	#MC	Machine Check	Error codes (if any) and source are model dependent. ⁴
19	#XM	SIMD Floating-Point Exception	SIMD Floating-Point Instruction ⁵
20-31		Reserved	
32-255		Maskable Interrupts	External interrupt from INTR pin or INT <i>n</i> instruction.

11. What is interrupt mask bit?

It is an internal switch setting that controls whether an interrupt can be processed or not. The mask is a bit that is turned on and off by the program. However it is also set automatically when you are in an ISR (interrupt service routine), at least for that interrupt, which is to allow the ISR to finish before taking the interrupt again. The interrupt mask is implemented in hardware and is part of the particular architecture of the CPU.

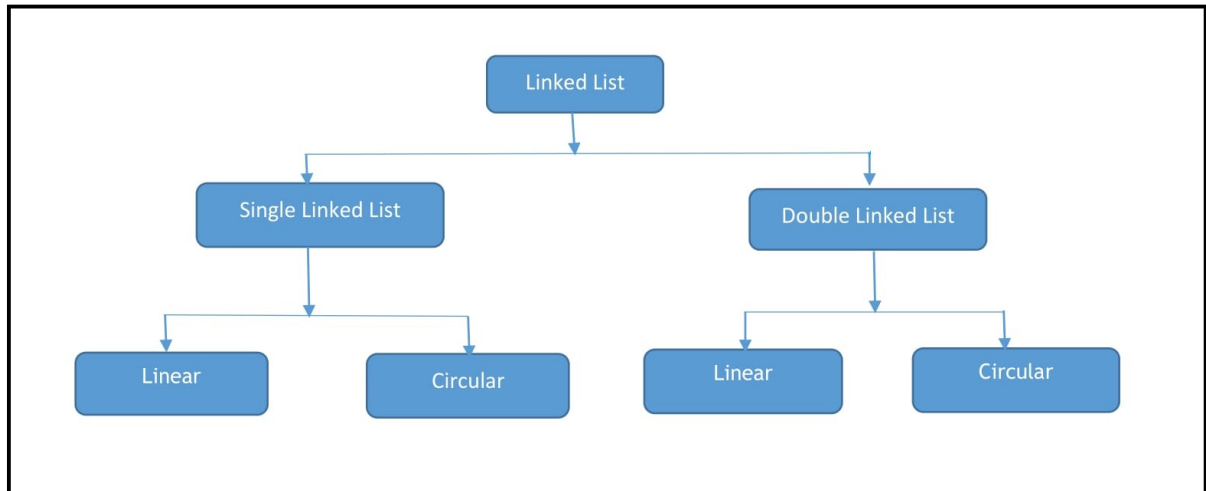
12. What are timers and counters? Are they different on hardware or are they same?

A timer is a specialized type of clock which is used to measure time intervals. A timer that counts from zero upwards for measuring time elapsed is often called a stopwatch. It is a device that counts down from a specified time interval and used to generate a time delay, for example, an hourglass is a timer.

A counter is a device that stores (and sometimes displays) the number of times a particular event or process occurred, with respect to a clock signal. It is used to count the events happening outside the microcontroller. In electronics, counters can be implemented quite easily using register-type circuits such as a flip-flop. The points that differentiate a timer from a counter are as follows:

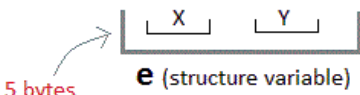
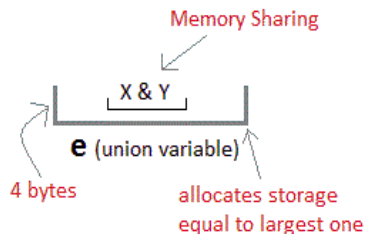
Timer	Counter
The register incremented for every machine cycle.	The register is incremented considering 1 to 0 transition at its corresponding to an external input pin (T0, T1).
Maximum count rate is 1/12 of the oscillator frequency.	Maximum count rate is 1/24 of the oscillator frequency.
A timer uses the frequency of the internal clock, and generates delay.	A counter uses an external signal to count pulses.

13. What are the different types of linked lists?



14. What is the difference between structures and unions?

	Structure	Union
Memory Allocation	Members of structure do not share memory. So a structure needs separate memory space for all its members i.e. all the members have unique storage.	A union shares the memory space among its members, so there is no need to allocate memory to all the members. Shared memory space is allocated i.e. equivalent to the size of member having largest memory.
Member Access	Members of structure can be accessed individually at any time.	At a time, only one member of union can be accessed.
Keyword	To define Structure, ' struct ' keyword is used.	To define Union, ' union ' keyword is used.
Initialization	All members of structure can be initialized.	Only the first member of Union can be initialized.
Size	Size of the structure is \geq to the sum of the each member's size.	Size of union is equivalent to the size of the member having largest size.
Syntax	struct struct_name	union union_name

	<pre>{ structure ele 1; structure ele 2; ----- ----- structure ele n; }struct_variable_name;</pre>	<pre>{ union ele 1; union ele 2; ----- ----- union ele n; }union_variable_name;</pre>
Change in Value	Change in the value of one member cannot affect the other in structure.	Change in the value of one member can affect the value of other member.
Storage in Memory	<p style="text-align: center;"><u>Structure</u></p> <pre>struct Emp { char X; // size 1 byte float Y; // size 4 byte }e;</pre> 	<p style="text-align: center;"><u>Unions</u></p> <pre>union Emp { char X; float Y; }e;</pre>  <p style="text-align: right;">allocates storage equal to largest one</p>

15. Explain bit-wise operators with example.

Operators	Bit-wise Operation	Bit-wise Assignment operators
& Bitwise AND operator	<p>Logical AND operation will be performed, when “&” is applied. For any given input, if any of the input bit is low, corresponding output bit will be low. Only if both input bits are high, output bit will be high.</p> <p>E.g.</p> <pre>10001000 10000001 & 10000000</pre>	&=
 Bitwise OR	<p>Logical OR operation will be performed, when “ ” is applied. For any given input, if any of the input bit is high,</p>	 =

operator	<p>corresponding output bit will be high. Only when both input bits are low, output bit will be low.</p> <p>E.g. 10001000 10000001 10001001</p>	
^ Bitwise XOR operator	<p>It is equivalent to adding two bits and discarding the carry. If two bits are not similar, output is high. If two bits are similar, output is low. XOR can be used to toggle bits between 1 and 0.</p> <p>E.g. 10001000 10000001 ^ 00001001</p>	^ =
~ Bitwise NOT operator	<p>Bit-wise NOT gives complement of given number. Simply, the bits are inverted.</p> <p>E.g. x = 10001000 ~x = 01110111 ~</p>	~ =
>> Right shift operator	<p>Right shift shifts each bit in its left operand to the right. The number followed by the operator decides the number of places the bits are shifted. Right shift is nothing but dividing a bit pattern by 2. Finally, the blanks are filled by zero.</p> <p>E.g. Empty bit locations should be filled depending upon the modifier associated with the basic integral data type.</p> <p><u>Case 1: signed char / signed int</u> When a number is shifted right, empty bit locations will be filled with the MSB bit of that particular number. In the below given example, MSB bit is 1. Hence, empty bit location is filled with 1. x = 11100111 >> 1 11110011</p> <p><u>Case-2: Unsigned char / Unsigned int</u> When a number is shifted right, empty bit locations will be always filled with the 0's only. x = 11100111 >> 1 01110011</p>	>> =
<< Left shift operator	<p>Left shift shifts each bit in its left operand to the left. Number followed by the operator decides the number of places the bits are shifted. Left shift is nothing but multiplying a bit pattern by 2. Finally, the blanks are filled by zero.</p> <p>E.g. x = 11100111 << 1</p>	<< =

	11001110	
--	----------	--

Refer link: <http://blog.emertxe.com/2017/09/bitwise-operators-in-c.html>

16. What are the different layers in OSI model?

layer 7 application	Applications and application interfaces for OSI networks. Provides access to lower layer functions and services.
layer 6 presentation	Negotiates syntactic representations and performs data transformations, e.g. compression and code conversion.
layer 5 session	Coordinates connection and interaction between applications, established dialog, manages and synchronizes data flow direction.
layer 4 transport	Ensures end-to-end data transfer and integrity across the network. Assembles packets for routing by Layer 3.
layer 3 network	Routes and relays data units across a network of nodes. Manages flow control and call establishment procedures.
layer 2 data link	Transfers data units from one network unit to another over transmission circuit. Ensures data integrity between nodes.
layer 1 physical	Delimits and encodes the bits onto the physical medium. Defines electrical, mechanical and procedural formats.

17. What is shell scripting?

A shell script is a computer program designed to be run by the Unix Shell, a command line interpreter. The various dialects of shell scripts are considered to be scripting languages. Typical operations performed by shell scripts include file manipulation, program execution, and printing text. A script which sets up the environment, runs the program, and does any necessary cleanup, logging, etc. is called a wrapper.

18. How do you obtain the outputs of different stages of compilation?

Refer Link: <https://www.calleerlandsson.com/the-four-stages-of-compiling-a-c-program/>

19. What is the niceness value of a thread?

The Linux niceness scale goes from **-20** to 19. The lower the number the more priority that task gets. If the niceness value is high number like 19 the task will be set to the lowest priority and the CPU will process it whenever it gets a chance. The default nice value is zero.

20. Why do we need IPC?

Interprocess communication (IPC) is a set of programming interfaces that allow a programmer to coordinate activities among different program processes that can run concurrently in an operating system.

This allows a program to handle many user requests at the same time. Since even a single user request may result in multiple processes running in the operating system on the user's behalf, the processes need to communicate with each other. The IPC interfaces make this possible. Each IPC method has its own advantages and limitations so it is not unusual for a single program to use all of the IPC methods.

IPC methods include pipes and named pipes, message queuing, semaphores, shared memory, and sockets.

21. Define a structure. How to access each member of the structure?

1. Array elements are accessed using the Subscript variable, Similarly Structure members are accessed using dot [.] operator.
2. (.) is called as "Structure member Operator".
3. Use this Operator in between "**Structure name**" & "**member name**"

```
#include<stdio.h>
struct Vehicle
{
    int wheels;
    char vname[20];
    char color[10];
}v1 = {4,"Nano","Red"};

int main()
{
    printf("Vehicle No of Wheels : %d\n",v1.wheels);
    printf("Vehicle Name : %s\n",v1.vname);
    printf("Vehicle Color : %s\n",v1.color);
    return(0);
}
```

22. What is padding? How to remove padding? Who manages padding?

1. In order to align the data in memory, one or more empty bytes (addresses) are inserted (or left empty) between memory addresses which are allocated for other

structure members during memory allocation. This concept is called structure padding.

2. Use “#pragma pack (1)” to disable/remove byte padding. Also, we can use __attribute__((packed, aligned(X))) to insist particular(X) sized padding. X should be powers of two.

3. Structure padding is managed by the respective compiler.

Refer Link: <https://fresh2refresh.com/c-programming/c-structure-padding/>

23. Can we simply rearrange the members of the structure to reduce padding?

We cannot reduce padding by re-arranging the members of the structure. We can always attempt to reduce padding by declaring the smallest data types last in the structure. To avoid padding there are compiler options like pragma pack.

```
struct _foo {  
    int a; /* No padding between a & b */  
    short b;  
} foo;  
  
struct _bar {  
    short b; /* 2 bytes of padding between a & b */  
    int a;  
} bar;
```

24. How will you create a thread?

Normally when a program starts up and becomes a process, it starts with a default thread. So we can say that every process has at least one thread of control. A process can create extra threads using the following function:

```
#include <pthread.h>  
int pthread_create(pthread_t *restrict tidp, const pthread_attr_t *restrict attr, void  
*(*start_rtn)(void), void *restrict arg).
```

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

```
void * thread1()
{
    while(1){
        printf("Hello!!\n");
    }
}
void * thread2()
{
    while(1){
        printf("How are you?\n");
    }
}
int main()
{
    int status;
    pthread_t tid1,tid2;
    pthread_create(&tid1,NULL,thread1,NULL);
    pthread_create(&tid2,NULL,thread2,NULL);
    pthread_join(tid1,NULL);
    pthread_join(tid2,NULL);
    return 0;
}
```

The above function requires four arguments, let's first discuss a bit on them:

- The first argument is a pthread_t type address. Once the function is called successfully, the variable whose address is passed as first argument will hold the thread ID of the newly created thread.
- The second argument may contain certain attributes which we want the new thread to contain. It could be priority etc.
- The third argument is a function pointer. This is something to keep in mind that each thread starts with a function and that function's address is passed here as the third argument so that the kernel knows which function to start the thread from.
- As the function (whose address is passed in the third argument above) may accept some arguments also so we can pass these arguments in form of a pointer to a void type. Now, why a void type was chosen? This was because if a function accepts more than one argument then this pointer could be a pointer to a structure that may contain these arguments.

25. How will you create a new process?

By using fork() system call, we can create new process.

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

int main ()
{
    printf("Fork Demonstration\n");
    fork();
    printf("\nHello World\n");
    return 0;
}
```

26. Write a program to obtain the greatest among 3 numbers.

```
#define LARGEST(a,b,c) (a>b?(a>c?a:c):(b>c?b:c))
#include <stdio.h>
#include <stdio.h>
int main()
{
    int a,b,c,Largest;
    printf("enter the numbers\n");
    scanf("%d %d %d",&a,&b,&c);
    Largest = LARGEST(a,b,c);
    printf("Largest number is %d",Largest);
}
```

27. Write a program to check if nth bit of a number is set or not.

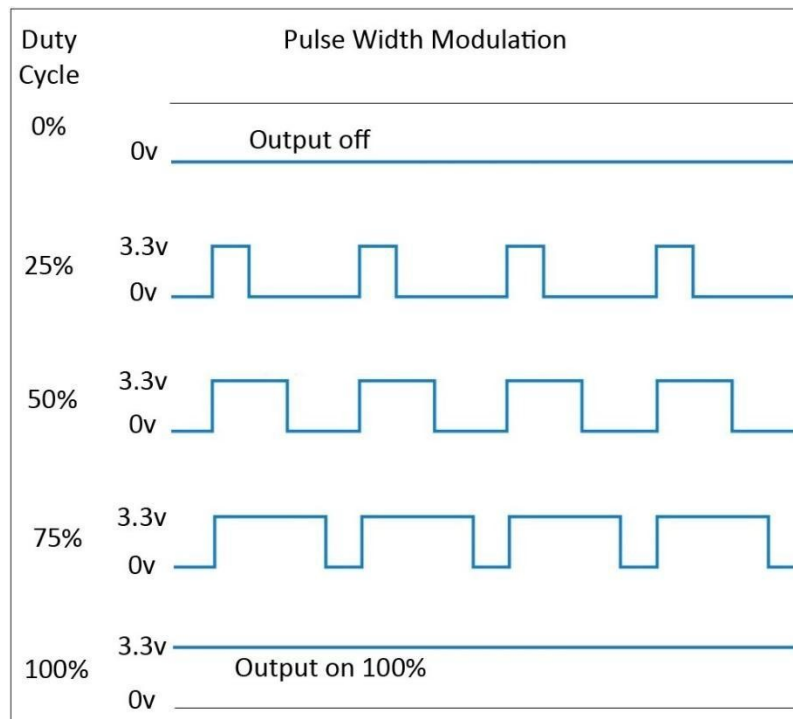
```
#define BITSET(NUM,N) (NUM & (1<<N))
#include <stdio.h>
int main()
{
    int NUM; //to store number
    int N; //to store bit
    int bit_set;
```

```
printf("Enter an 8 bits integer number: ");
scanf("%d",&NUM);
printf("Now, enter a bit number (from 0 to 7) to check, whether it is SET or not:
");
scanf("%d",&N);

bit_set = BITSET(NUM,N); //checking bit status
if(bit_set)
    printf("Bit number %d is SET in number %d.\n",N,NUM);
else
    printf("Bit number %d is not SET in number %d.\n",N,NUM);
return 0;
}
```

28. Explain PWM. Draw waveforms for 100% and 25% duty cycle

Pulse-width modulation (PWM), or pulse-duration modulation (PDM), is a modulation technique used to encode a message into a pulsing signal. Although this modulation technique can be used to encode information for transmission, its main use is to allow the control of the power supplied to electrical devices, especially to inertial loads such as motors.



29. How data or message is transferred from one process to another process?
Data or message is transferred from one process to another process through inter process mechanisms.

30. How do you say whether a microcontroller is 8bit or 16bit?
The best way is to read the data sheet of the Microcontroller.

31. Explain both pre and post - increment and decrement operations.

Operator	Operator/Description
Pre increment operator (++i)	Consider $x = ++i$; At First, i is incremented. ($i=i+1$) Then, value of i is assigned to x ($x = i$).
Post increment operator (i++)	Consider $x = i++$; At First, Value of i is assigned to x ($x = i$). Then, i is incremented ($i=i+1$)
Pre decrement operator (--i)	Consider $x = --i$; At First, i is decremented. ($i=i-1$) Then, value of i is assigned to x ($x = i$).
Post decrement operator (i--)	Consider $x = i--$; At First, Value of i is assigned to x ($x = i$). Then, i is decremented ($i=i-1$)

32. Write programs to explain Static, extern and Global variables.

Static Variables

A static int variable remains in memory while the program is running. A normal or auto variable is destroyed when a function call where the variable was declared is over. For example, we can use static int to count number of times a function is called, but an auto variable can't be used for this purpose.

```
#include<stdio.h>
int fun()
{
    static int count = 0;
    count++;
    return count;
}
int main()
{
    printf("%d ", fun());
    printf("%d ", fun());
    return 0;
}
```

External (extern) storage class:

1. Variables of this storage class are "Global variables"
 2. Global Variables are declared outside the function and are accessible to all functions in the program
 3. Generally , External variables are declared again in the function using keyword extern
 4. In order to Explicit declaration of variable use 'extern' keyword
- extern int num1; // Explicit Declaration.

Example:

```
int num = 75 ;
void display();
void main()
{
    extern int num ;
    printf("Num : %d",num);
    display();
}
void display()
{
    extern int num ;
    printf("Num : %d",num);
}
```

Global variable:

The scope of global variables will be throughout the program. These variables can be accessed from anywhere in the program. This variable is defined outside the main function. So that, this variable is visible to main function and all other sub functions.

```
include<stdio.h>
void test();int m = 22, n = 44;
int a = 50, b = 80;

int main()
{
    printf("All variables are accessed from main function");
    printf("\nvalues: m=%d:n=%d:a=%d:b=%d", m,n,a,b);
    test();
}

void test()
{
    printf("\n\nAll variables are accessed from" \
    " test function");
    printf("\nvalues: m=%d:n=%d:a=%d:b=%d", m,n,a,b);
}
```

OUTPUT:

```
All variables are accessed from main function
values : m = 22 : n = 44 : a = 50 : b = 80
All variables are accessed from test function
values : m = 22 : n = 44 : a = 50 : b = 80
```

	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

33. What is interrupt? Explain. If interrupt occurs, how system will perform?

An interrupt is a signal to the processor emitted by hardware or software indicating an event that needs immediate attention. An interrupt alerts the processor to a high-priority condition requiring the interruption of the current code the processor is executing. The processor responds by suspending its current activities, saving its state, and executing a function called an interrupt handler (or an interrupt service routine, ISR) to deal with the event.

34. How concurrency is achieved in threads?

The Linux kernel and the pThreads libraries work together to administer the threads. The kernel does the context switching, scheduling, memory management, cache memory management, etc. There is other administration done at the user level also. The kernel treats each process-thread as one entity. It has its own rules about time slicing that take processes (and process priorities) into consideration but each sub-process thread is a schedulable entity.

35. If you create a new process and you have declared a global variable $a = 3$, if you change the value of 'a' in child process, will it get change in parent process?

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

```
#include <sys/types.h>
#include<unistd.h>
int a = 3;
int main()
{
    // child process because return value zero
    if (fork()==0)
    {
        a = 30;
        printf("Hello from Child!\n");
    }
    // parent process because return value non-zero.
    else
    {
        printf("Hello from Parent!\n");
        printf("value of a is %d\n",a);
    }
}
```

Output:

```
Hello from Parent!
Hello from Child!
value of a in parent process 3
```

If you create a new process and have declared a global variable `a = 3`, and if the value of 'a' is changed in child process, it won't get changed in the parent process.

36. What is array of structures and how value is assigned to that array of structure and also by using pointer assign value to that array of structure?

Structure is collection of different datatypes (variables) which are grouped together. Whereas, array of structures is nothing but collection of structures. This is also called as structure array in C.

```
#include <stdio.h>
#include <string.h>

struct student
{
int id;
char name[30];
float percentage;
};

int main()
{
int i;
struct student record[2];

// 1st student's record
record[0].id=1;
strcpy(record[0].name, "Raju");
record[0].percentage = 86.5;

// 2nd student's record
record[1].id=2;
strcpy(record[1].name, "Surendren");
record[1].percentage = 90.5;

// 3rd student's record
record[2].id=3;
strcpy(record[2].name, "Thiyagu");
record[2].percentage = 81.5;

for(i=0; i<3; i++)
{
printf("    Records of STUDENT : %d \n", i+1);
printf(" Id is: %d \n", record[i].id);
printf(" Name is: %s \n", record[i].name);
printf(" Percentage is: %f\n\n",record[i].percentage);
}
return 0;
}
```

OUTPUT:

Records of STUDENT : 1

Id is: 1

Name is: Raju

Percentage is: 86.500000
Records of STUDENT : 2
Id is: 2
Name is: Surendren
Percentage is: 90.500000
Records of STUDENT : 3
Id is: 3
Name is: Thiyagu
Percentage is: 81.500000

Using pointer:

```
#include <stdio.h>
#include <string.h>

struct student
{
    int id;
    char name[30];
    float percentage;
};

int main()
{
    int i;
    struct student record1 = {1, "Raju", 90.5};
    struct student *ptr;

    ptr = &record1;

    printf("Records of STUDENT1: \n");
    printf(" Id is: %d \n", ptr->id);
    printf(" Name is: %s \n", ptr->name);
    printf(" Percentage is: %f \n\n", ptr->percentage);

    return 0;
}
```

OUTPUT:

Records of STUDENT1:
Id is: 1
Name is: Raju
Percentage is: 90.500000

37. Which is better? Using threads on single core or multicore processor?

Using threads on multi-core processor is better since we achieve parallelism. Find the reasons below.

Single-core processor:

Multi-threading:

The threads will execute concurrently, even on a single core. This is because of preemptive scheduling, which means that the OS will pause threads to let other threads run.

This is true for all the common desktop OS.

Multi-core processor:

Multi-threading:

In computer architecture, multithreading is the ability of a central processing unit (CPU) to execute multiple processes or threads concurrently, appropriately supported by the operating system.

Parallelism:

Parallelism is a general technique of using more than one flow of instructions to complete a computation. The critical aspect of all parallel techniques is communicating between flows to collaborate a final answer. Threading is a specific implementation of parallelism. Each flow of instructions is given its own stack to keep a record of local variables and function calls, and communicates with the other flows implicitly by shared memory. One example might be to have one thread simply queue up disk requests and pass it to a worker thread, effectively parallelizing disk and CPU.

You can have multithreading on a single core machine as well as multi core machine, but you can only have parallelism only on a multi core machine.

38. Explain 8086 microprocessors.

- How many bit processor it is?
- Why is it 16 bit?
- Can we do 32-bit operations?

8086 Microprocessor is an enhanced version of 8085 Microprocessor that was designed by Intel in 1976. It is a 16-bit Microprocessor having 20 address lines and 16 data lines that provides up to 1MB storage. It consists of powerful instruction set, which provides operations like multiplication and division easily. It supports two modes of operation, i.e. Maximum mode and Minimum mode. Maximum mode is suitable for system having multiple processors and Minimum mode is suitable for system having a single processor.

Features of 8086

The most prominent features of a 8086 microprocessor are as follows –

- It has an instruction queue, which is capable of storing six instruction bytes from the memory resulting in faster processing.
- It was the first 16-bit processor having 16-bit ALU, 16-bit registers, internal data bus, and 16-bit external data bus resulting in faster processing.
- It is available in 3 versions based on the frequency of operation –
 - o 8086 → 5MHz
 - o 8086-2 → 8MHz
 - o (c)8086-1 → 10 MHz

- It uses two stages of pipelining, i.e. Fetch Stage and Execute Stage, which improves performance.
- Fetch stage can prefetch up to 6 bytes of instructions and stores them in the queue.
- Execute stage executes these instructions.
- It has 256 vectored interrupts.
- It consists of 29,000 transistors.

39. Write a function 1 in which function 2 is called for every 3 seconds and function 3 is called for every 8 seconds.

```
Void interrupt timer_isr(void)
{
    /* Init timer register for 1second overflow */
    if( sec1++ == 2)
    {
        sec1 = 0;
        fn2();
    }
    if( sec2++ == 8 )
    {
        sec2 = 0;
        fn3();
    }
}
```

40. What are the different compiler flags that you know?

- O for computer optimizations, can add 0,1,2,3 to it, to control optimization levels.
- Wall Enables all warnings
- Werror to treat warnings as errors
- E to generate preprocessed output
- v to see the version of compiler

Note: refer “man gcc” for more flags

41. What happens when interrupt flag is raised in microcontroller?

The control gets transferred to the IVT (Interrupt Vector Table) or ISR(Interrupt Service Routine). This depends on the compiler designed for different architecture.

42. How to generate an interrupt for every 5000 ticks?

This can be done by using timer. Consider the below example – PIC microcontroller with 16 bit timer.

Timer Resolution – 16 bits

Timer Quantum – 1 micro second (1tick = 1 micro second)

Prescaler value – 1:1

Load the timer register with = $2^{16} - 1 - 5000$
= 65536 – 1 – 5000
= 60535.

43. How do you write the ISR when interrupts from multiple sources occur?

This depends on architecture. If the controller is based on IVT, you would have different ISR pointed by the IVT. If you go with controllers like PIC, you might have one common interrupt vector and write code for all interrupts in it by checking a flag with that block.

E.g. For PIC microcontroller:

```
void interrupt_isr(void)
{
    if(T0IF)
    {
        /* Some code here */
    }
    if(T1IF)
    {
        /* Some code here */
    }
}
```

44. Describe college project.

45. What controller have you worked on? Explain.