Q.Who is the writer of c language?

Dennis Ritchie

The C Programming Language (sometimes termed K&R, after its authors' initials) is a computer programming book written by **Brian Kernighan** and **Dennis Ritchie**, the latter of whom originally designed and implemented the language, as well as co-designed the Unix operating system with which development of the language was closely intertwined. The book was central to the development and popularization of the C programming language and is still widely read and used today. Because the book was co-authored by the original language designer, and because the first edition of the book served for many years as the *de facto* standard for the language, the book was regarded by many to be the authoritative reference on C.

Q .In PC where c program stored & Run.

Q.Diff btw sdram and dram.

Summary:

1. SRAM is static while DRAM is dynamic
2. SRAM is faster compared to DRAM
3. SRAM consumes less power than DRAM
4. SRAM uses more transistors per bit of memory compared to DRAM
5. SRAM is more expensive than DRAM
6. Cheaper DRAM is used in main memory while SRAM is commonly used in cache memory

Q.Types of ROM.

Key Difference: ROM stands for Read Only Memory. This is a non-volatile type of memory which is found in computers and other electronics devices. It is used for storing data permanently. Some of the common types of ROM include – Mask ROM, PROM, EPROM and EEPROM.

Mask ROM – It is also known as MROM. It is a static ROM which comes programmed into an integrated circuit by its manufacturer. The software mask is burned into the chip and that is why it is termed as Mask ROM. Network operating systems and server operating systems make use of this type of Mask ROM chips.

PROM –It stands for Programmable Read Only Memory. It was first developed in 70s by Texas Instruments. It is made as a blank memory. A PROM programmer or PROM burner is required in order to write data onto a PROM chip. The data stored in it cannot be modified and therefore it is also known as one time programmable device

EPROM – It stands for Erasable Programmable ROM. It is different from PROM as unlike PROM the program can be written on it more than once. This comes as the solution to the problem faced by PROM. The bits of memory come back to 1, when ultra violet rays of some specific wavelength falls into its chip’s glass panel. The fuses are reconstituted and thus new things can be written on the memory

EEPROM- It stands for Electrically Erasable Read Only Memory. These are also erasable like EPROM, but the same work of erasing is performed with electric current. Thus, it provides the ease of erasing it even if the memory is positioned in the computer. It stores computer system’s BIOS. Unlike EPROM, the entire chip does not have to be erased for changing some portion of it. Thus, it even gets rid of some biggest challenges faced by using EPROMs.

FLASH ROM– It is an updated version of EEPROM. In EEPROM, it is not possible to alter many memory locations at the same time. However, Flash memory provides this advantage over the EEPROM by enabling this feature of altering many locations simultaneously. It was

invented by Toshiba and got its name from it capability of deleting a block of data in a flash.

Q.What is Round robin scheduling.

CPU Scheduling

CPU scheduling is a process which allows one process to use the CPU while the execution of another process is on hold(in waiting state) due to unavailability of any resource like I/O etc, thereby making full use of CPU. The aim of CPU scheduling is to make the system efficient, fast and fair.

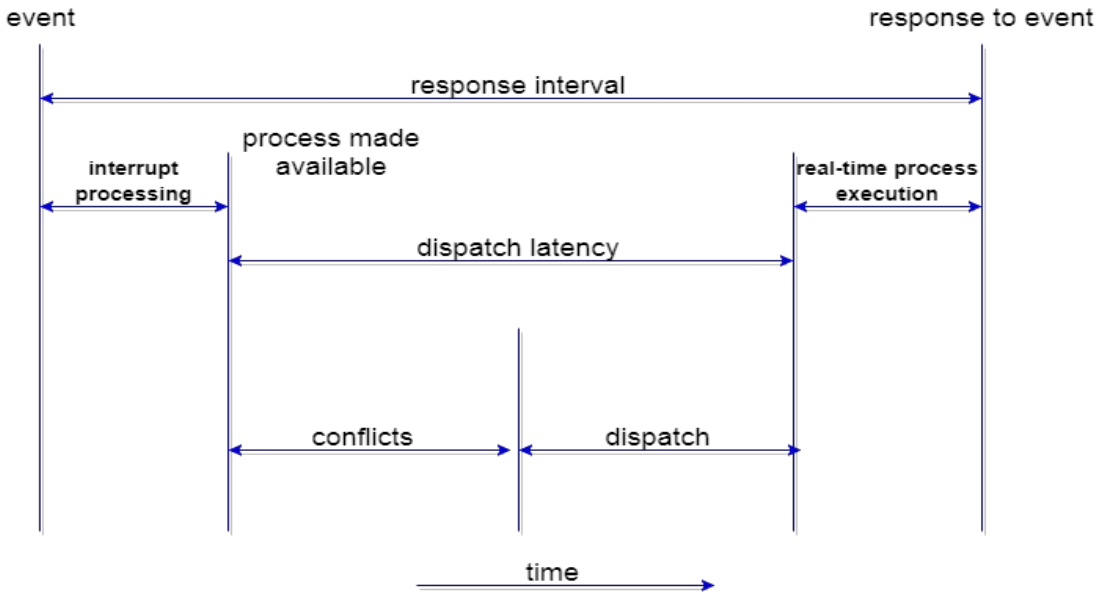
Whenever the CPU becomes idle, the operating system must select one of the processes in the ready queue to be executed. The selection process is carried out by the short-term scheduler (or CPU scheduler). The scheduler selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.

Dispatcher

Another component involved in the CPU scheduling function is the **dispatcher**. The dispatcher is the module that gives control of the CPU to the process selected by the **short-term scheduler**. This function involves:

- Switching context
- Switching to user mode
- Jumping to the proper location in the user program to restart that program

The dispatcher should be as fast as possible, given that it is invoked during every process switch. The time it takes for the dispatcher to stop one process and start another running is known as the **dispatch latency**. Dispatch Latency can be explained using the below figure:



Types of CPU Scheduling

CPU scheduling decisions may take place under the following four circumstances:

1. When a process switches from the **running** state to the **waiting** state(for I/O request or invocation of wait for the termination of one of the child processes).
2. When a process switches from the **running** state to the **ready** state (for example, when an interrupt occurs).
3. When a process switches from the **waiting** state to the **ready** state(for example, completion of I/O).
4. When a process **terminates**.

In circumstances 1 and 4, there is no choice in terms of scheduling. A new process(if one exists in the ready queue) must be selected for execution. There is a choice, however in circumstances 2 and 3.

When Scheduling takes place only under circumstances 1 and 4, we say the scheduling scheme is **non-preemptive**; otherwise the scheduling scheme is **preemptive**.

Non-Preemptive Scheduling

Under non-preemptive scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state.

This scheduling method is used by the Microsoft Windows 3.1 and by the Apple Macintosh operating systems.

It is the only method that can be used on certain hardware platforms, because It does not require the special hardware(for example: a timer) needed for preemptive scheduling.

Preemptive Scheduling

In this type of Scheduling, the tasks are usually assigned with priorities. At times it is necessary to run a certain task that has a higher priority before another task although it is running. Therefore, the running task is interrupted for some time and resumed later when the priority task has finished its execution.

Scheduling Criteria

There are many different criterias to check when considering the "best" scheduling algorithm :

- **CPU utilization**

  To make out the best use of CPU and not to waste any CPU cycle, CPU would be working most of the time(Ideally 100% of the time). Considering a real system, CPU usage should range from 40% (lightly loaded) to 90% (heavily loaded.)

- **Throughput**

  It is the total number of processes completed per unit time or rather say total amount of work

done in a unit of time. This may range from 10/second to 1/hour depending on the specific processes.

- **Turnaround time**

  It is the amount of time taken to execute a particular process, i.e. The interval from time of submission of the process to the time of completion of the process(Wall clock time).

- **Waiting time**

  The sum of the periods spent waiting in the ready queue amount of time a process has been waiting in the ready queue to acquire get control on the CPU.

- **Load average**

  It is the average number of processes residing in the ready queue waiting for their turn to get into the CPU.

- **Response time**

  Amount of time it takes from when a request was submitted until the first response is produced. Remember, it is the time till the first response and not the completion of process execution(final response).

In general CPU utilization and Throughput are maximized and other factors are reduced for proper optimization.

Scheduling Algorithms

We'll discuss four major scheduling algorithms here which are following :

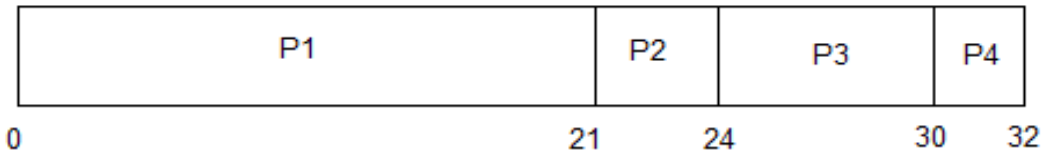
1. First Come First Serve(FCFS) Scheduling
2. Shortest-Job-First(SJF) Scheduling
3. Priority Scheduling
4. Round Robin(RR) Scheduling
5. Multilevel Queue Scheduling
6. Multilevel Feedback Queue Scheduling

First Come First Serve(FCFS) Scheduling

- Jobs are executed on first come, first serve basis.
- Easy to understand and implement.
- Poor in performance as average wait time is high.

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

The average waiting time will be = (0 + 21 + 24 + 30)/4 = 18.75 ms



This is the GANTT chart for the above processes

Shortest-Job-First(SJF) Scheduling

- Best approach to minimize waiting time.
- Actual time taken by the process is already known to processor.
- Impossible to implement.

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

In Shortest Job First Scheduling, the shortest Process is executed first.

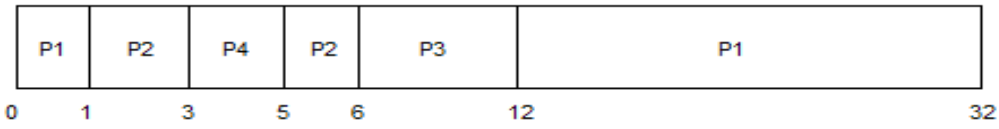
Hence the GANTT chart will be following :

Now, the average waiting time will be = (0 + 2 + 5 + 11)/4 = 4.5 ms

In Preemptive Shortest Job First Scheduling, jobs are put into ready queue as they arrive, but as a process with short burst time arrives, the existing process is preempted.

PROCESS	BURST TIME	ARRIVAL TIME
P1	21	0
P2	3	1
P3	6	2
P4	2	3

The GANTT chart for Preemptive Shortest Job First Scheduling will be,



The average waiting time will be, ((5-3) + (6-2) + (12-1))/4 = 4.25 ms

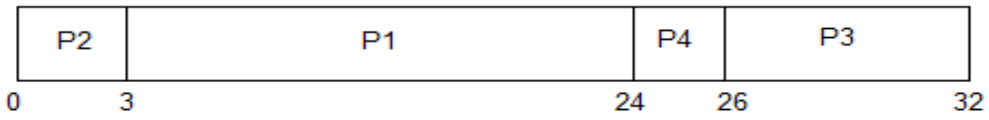
The average waiting time for preemptive shortest job first scheduling is less than both, non-preemptive SJF scheduling and FCFS scheduling.

Priority Scheduling

- Priority is assigned for each process.
- Process with highest priority is executed first and so on.
- Processes with same priority are executed in FCFS manner.
- Priority can be decided based on memory requirements, time requirements or any other resource requirement.

PROCESS	BURST TIME	PRIORITY
P1	21	2
P2	3	1
P3	6	4
P4	2	3

The GANTT chart for following processes based on Priority scheduling will be,



The average waiting time will be, (0 + 3 + 24 + 26)/4 = 13.25 ms

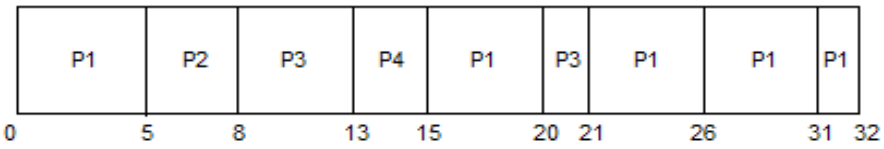
---

Round Robin(RR) Scheduling

- A fixed time is allotted to each process, called **quantum**, for execution.
- Once a process is executed for given time period that process is preemptied and other process executes for given time period.
- Context switching is used to save states of preemptied processes.

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

The GANTT chart for round robin scheduling will be,



The average waiting time will be, 11 ms.

Multilevel Queue Scheduling

Another class of scheduling algorithms has been created for situations in which processes are easily classified into different groups.

For example: A common division is made between foreground(or interactive) processes and background (or batch) processes. These two types of processes have different response-time requirements, and so might have different scheduling needs. In addition, foreground processes may have priority over background processes.

A multi-level queue scheduling algorithm partitions the ready queue into several separate queues. The processes are permanently assigned to one queue, generally based on some property of the process, such as memory size, process priority, or process type. Each queue has its own scheduling algorithm.

For example: separate queues might be used for foreground and background processes. The foreground queue might be scheduled by Round Robin algorithm, while the background queue is scheduled by an FCFS algorithm.

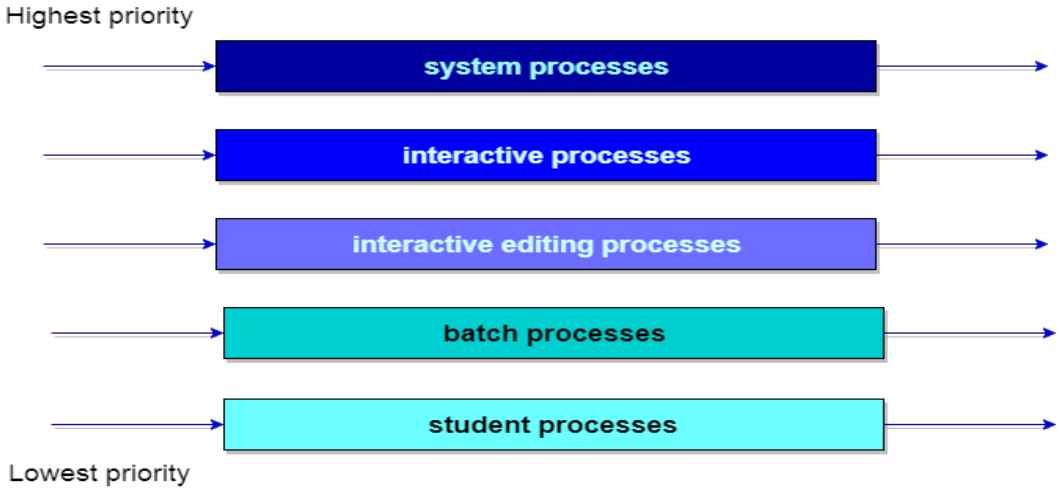
In addition, there must be scheduling among the queues, which is commonly implemented as fixed-priority preemptive scheduling. **For example:** The foreground queue may have absolute priority over the background queue.

Let us consider an example of a multilevel queue-scheduling algorithm with five queues:

1. System Processes

2. Interactive Processes
3. Interactive Editing Processes
4. Batch Processes
5. Student Processes

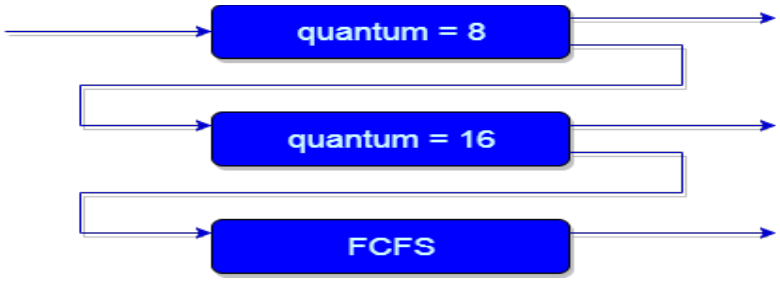
Each queue has absolute priority over lower-priority queues. No process in the batch queue, for example, could run unless the queues for system processes, interactive processes, and interactive editing processes were all empty. If an interactive editing process entered the ready queue while a batch process was running, the batch process will be preempted.



Multilevel Feedback Queue Scheduling

In a multilevel queue-scheduling algorithm, processes are permanently assigned to a queue on entry to the system. Processes do not move between queues. This setup has the advantage of low scheduling overhead, but the disadvantage of being inflexible.

Multilevel feedback queue scheduling, however, allows a process to move between queues. The idea is to separate processes with different CPU-burst characteristics. If a process uses too much CPU time, it will be moved to a lower-priority queue. Similarly, a process that waits too long in a lower-priority queue may be moved to a higher-priority queue. This form of aging prevents starvation.



An example of a multilevel feedback queue can be seen in the below figure.

In general, a multilevel feedback queue scheduler is defined by the following parameters:

- The number of queues.
- The scheduling algorithm for each queue.
- The method used to determine when to upgrade a process to a higher-priority queue.
- The method used to determine when to demote a process to a lower-priority queue.
- The method used to determine which queue a process will enter when that process needs service.

The definition of a multilevel feedback queue scheduler makes it the most general CPU-scheduling algorithm. It can be configured to match a specific system under design. Unfortunately, it also requires some means of selecting values for all the parameters to define the best scheduler. Although a multilevel feedback queue is the **most general scheme**, it is also the **most complex**

Q.**Which is the default scheduling algorithm in PC. What happened when PC start(Boot process).**

About i2c.... Spi and i2c Diff.

Q.What is meant by Volatile non volatile memory.

Volatile Memory :

A device which holds the data as long as it has power supply connected to it and cannot hold the memory when there is no power supply connected to it is called Volatile Memory. The best example for this can be Random Access Memory (RAM), which will hold memory only as long as it is connected to power source and everything in it will be cleared if it gets disconnected from power source. Volatile memory is also called as temporary memory as it will hold memory temporarily.

Non Volatile Memory :

A device which can hold data in it even if it is not connected to any power source is called Non Volatile Memory. The typical examples for Non Volatile Memory are your Hard drives and flash drives. Even if you turn off your PC the data in your hard drive or flash drive stays intact.

Q.**C language is inter priter or compile based Lang**?

C as an intermediate language.

Q.What are Storage classes .

Q.Compilation steps of c prog.

A C’s program building process involves four stages and utilizes different ‘tools’ such as a preprocessor, compiler, assembler, and linker.

At the end there should be a single executable file. Given below are the stages that happen in order regardless of the operating system and the compiler we use.

1. ***Preprocessing*** is the first pass of any C compilation. It processes include-files, conditional compilation instructions and macros.
2. ***Compilation*** is the second pass. It takes the output of the preprocessor, and the source code, and generates assembler source code.
3. ***Assembly*** is the third stage of compilation. It takes the assembly source code and produces an assembly listing with offsets. The assembler output is stored in an object file.
4. ***Linking*** is the final stage of compilation. It takes one or more object files or libraries as input and combines them to produce a single (usually executable) file. In doing so, it resolves references to external symbols, assigns final addresses to procedures/functions and variables, and revises code and data to reflect new addresses (a process called relocation).

**if you use the IDE type compilers, these processes are quite transparent.*

Now we are going to examine more details about the process that happen before and after the linking stage. For any given input file, the file name suffix (file extension) determines what kind of compilation is done.

```
In UNIX/Linux, the executable or binary file doesn't have extension whereas in
Windows the executables for example may have .exe, .com and .dll.
```

Here are a few,

- file_name.c C source code which must be preprocessed.
- file_name.h C header file (not to be compiled or linked).
- file_name.C C++ source code which must be preprocessed.

OBJECT FILES and EXECUTABLE

After the source code has been assembled, it will produce an Object files (e.g. .o, .obj) and then linked, producing an executable files.

An object and executable come in several formats such as ELF (Executable and Linking Format) and COFF (Common Object-File Format). For example, ELF is used on Linux systems, while COFF is used on Windows systems.

A few other file formats are

- a.out The a.out format is the original file format for Unix.
- COFF The COFF (Common Object File Format) action is limited.

  When we examine the content of these object files there are areas called sections. Sections can hold executable code, data, dynamic linking information, debugging data, symbol tables, relocation information, comments, string tables, and notes.

Some sections are loaded into the process image and some provide information needed in the

building of a process image while still others are used only in linking object files.

For your quick reference :

Q.user and kernel specs Diff.

Kernel Mode

In Kernel mode, the executing code has complete and unrestricted access to the underlying hardware. It can execute any CPU instruction and reference any memory address. Kernel mode is generally reserved for the lowest-level, most trusted functions of the operating system. Crashes in kernel mode are catastrophic; they will halt the entire PC.

- **User Mode**

In User mode, the executing code has no ability to directly access hardware or reference memory. Code running in user mode must delegate to system APIs to access hardware or memory. Due to the protection afforded by this sort of isolation, crashes in user mode are always recoverable. Most of the code running on your computer will execute in user mode

Q.What is volatile and const qualifiers?

`const` means that the variable cannot be modified by the c code, not that it cannot change. It means that no instruction can write to the variable, but its value might still change.

`volatile` means that the variable may change at any time and thus no cached values might be used; each access to the variable has to be executed to its memory address.

C prog to swap a number without temp.

In Single link list add at begin and delete last node.

Implement atoi,atol,

*int atoi64(char *nptr) - Convert string to int64
*int atoi(char *nptr) - Convert string to int
*long atol(char *nptr) - Convert string to long

What is Dynamic memory allocation?

Dynamic Memory Allocation

- It is a process of allocating or de-allocating the memory at run time it is called as dynamically memory allocation.
- When we are working with array or string static memory allocation will be take place that is compile time memory management.
- When we ate allocating the memory at compile we cannot extend the memory at run time, if it is not sufficient.
- By using compile time memory management we cannot utilize the memory properly
- In implementation when we need to utilize the memory more efficiently then go for dynamic

memory allocation.

- By using dynamic memory allocation whenever we want which type we want or how much we type that time and size and that we much create dynamically.

Dynamic memory allocation related all predefined functions are declared in following header files.

- <alloc.h>
- <malloc.h>
- <mem.h>
- <stdlib.h>

Dynamic memory allocation related functions

Malloc()

By using malloc() we can create the memory dynamically at initial stage. Malloc() required one argument of type size type that is data type size malloc() will creates the memory in bytes format. Malloc() through created memory initial value is garbage.

Syntax:

```
Void*malloc(size type);
```

Note: Dynamic memory allocation related function can be applied for any data type that's why dynamic memory allocation related functions return void*.

When we are working with dynamic memory allocation type specification will be available at the time of execution that's why we required to use type casting process.

Syntax

```
int *ptr;
ptr=(int*)malloc(sizeof (int)); //2 byte

long double*ldptr;
ldptr=(long double*)malloc(sizeof(long double)) // 2 byte

char*cptr;
cptr=(char*)malloc(sizeof(char));       //1 byte

int*arr;
arr=(int*)malloc(sizeof int()*10);      //20 byte

cahr*str;
str=(char*)malloc(sizeof(char)*50);     //50 byte
```

calloc()

- By using calloc() we can create the memory dynamically at initial stage.
- calloc() required 2 arguments of type count, size-type.

- Count will provide number of elements; size-type is data type size.
- calloc() will creates the memory in blocks format.
- Initial value of the memory is zero.

Syntax

```
int*arr;
arr=(int*)calloc(10, sizeof(int));      // 20 byte
cahr*str;
str=(char*)calloc(50, siceof(char));    // 50 byte
```

realloc()

- By using realloc() we can create the memory dynamically at middle stage.
- Generally by using realloc() we can reallocation the memory.
- Realloc() required 2 arguments of type void*, size_type.
- Void* will indicates previous block base address, size-type is data type size.
- Realloc() will creates the memory in bytes format and initial value is garbage.

Syntax

```
void*realloc(void*, size-type);
int *arr;
arr=(int*)calloc(5, sizeof(int));

arr=(int*)realloc(arr,sizeof(int)*10);
```

free()

- When we are working with dynamic memory allocation memory will created in heap area of data segment.
- When we are working with dynamic memory allocation related memory it is a permanent memory if we are not de-allocated that's why when we are working with dynamic memory allocation related program, then always recommended to deleted the memory at the end of the program.
- By using free(0 dynamic allocation memory can be de-allocated.
- free() requires one arguments of type void*.

Syntax

```
void free(voie*);
int *arr;
arr=(int*)calloc(10,sizeof(int));
free(arr);
```

- By using malloc(), calloc(), realloc() we can create maximum of 64kb data only.
- In implementation when we need to create more than 64kb data then go for formalloc(),

forcalloc() and forrealloc().

- By using free() we can de-allocate 64kb data only, if we need to de-allocate more than 64kb data then go for

Syntax

```
forfree().
formalloc()
voidfor*formalloc(size-type);
```

Q. Diff betw malloc and calloc.

Q. **How interrupt works? How to handle interrupt**?

What is an Interrupt?

Interrupt is a signal which has highest priority from hardware or software which processor should process its signal immediately.

Types of Interrupts:

Although interrupts have highest priority than other signals, there are many type of interrupts but basic type of interrupts are

1. **Hardware Interrupts:** If the signal for the processor is from external device or hardware is called hardware interrupts. Example: from keyboard we will press the key to do some action this pressing of key in keyboard will generate a signal which is given to the processor to do action, such interrupts are called hardware interrupts. Hardware interrupts can be classified into two types they are
    - **Maskable Interrupt:** The hardware interrupts which can be delayed when a much highest priority interrupt has occurred to the processor.
    - **Non Maskable Interrupt:** The hardware which cannot be delayed and should process by the processor immediately.
2. **Software Interrupts:** Software interrupt can also divided in to two types. They are
    - **Normal Interrupts:** the interrupts which are caused by the software instructions are called software instructions.
    - **Exception:** unplanned interrupts while executing a program is called Exception. For example: while executing a program if we got a value which should be divided by zero is called a exception.

Classification of Interrupts According to Periodicity of Occurrence:

1. **Periodic Interrupt:** If the interrupts occurred at fixed interval in timeline then that interrupts are called periodic interrupts
2. **Aperiodic Interrupt:** If the occurrence of interrupt cannot be predicted then that interrupt is called aperiodic interrupt.

Classification of Interrupts According to the Temporal Relationship with System Clock:

1. **Synchronous Interrupt:** The source of interrupt is in phase to the system clock is called synchronous interrupt. In other words interrupts which are dependent on the system clock. Example: timer service that uses the system clock.
2. **Asynchronous Interrupts:** If the interrupts are independent or not in phase to the system clock is called asynchronous interrupt.

Interrupt Handling:

We know that instruction cycle consists of fetch, decode, execute and read/write functions. After every instruction cycle the processor will check for interrupts to be processed if there is no interrupt is present in the system it will go for the next instruction cycle which is given by the instruction register.

If there is an interrupt present then it will trigger the interrupt handler, the handler will stop the present instruction which is processing and save its configuration in a register and load the program counter of the interrupt from a location which is given by the interrupt vector table. After processing the interrupt by the processor interrupt handler will load the instruction and its configuration from the saved register, process will start its processing where it's left. This saving the old instruction processing configuration and loading the new interrupt configuration is also called as context switching.

The interrupt handler is also called as Interrupt service routine (ISR). There are different types of interrupt handler which will handle different interrupts. For example for the clock in a system will have its interrupt handler, keyboard it will have its interrupt handler for every device it will have its interrupt handler.

The main features of the ISR are

- Interrupts can occur at any time they are asynchronous. ISR's can call for asynchronous interrupts.
- Interrupt service mechanism can call the ISR's from multiple sources.
- ISR's can handle both maskable and non maskable interrupts. An instruction in a program can disable or enable an interrupt handler call.
- ISR on beginning of execution it will disable other devices interrupt services. After completion of the ISR execution it will re initialize the interrupt services.
- The nested interrupts are allowed in ISR for diversion to other ISR.

Type of Interrupt Handlers:

1. First Level Interrupt Handler (FLIH) is hard interrupt handler or fast interrupt handler. These interrupt handlers have more jitter while process execution and they are mainly maskable interrupts
2. Second Level Interrupt Handler (SLIH) is soft interrupt handler and slow interrupt handler. These interrupt handlers are having less jitter.

Interrupt Latency:

When an interrupt occur, the service of the interrupt by executing the ISR may not start immediately by context switching. The time interval between the occurrence of interrupt and start of execution of

the ISR is called interrupt latency.

- **Tswitch** = Time taken for context switch
- **ΣTexec** = The sum of time interval for executing the ISR
- **Interrupt Latency** = Tswitch + ΣTexec

Q.What is the Use of gpio pins?

The reason we need pins that provide general purpose use is to provide an interface that can be controlled by various devices and similarly be used to control the behavior of other devices. As an example, a USB or Serial I/ O interface can be programmed to control lines on the GPIO though register setup, these in turn can be programmed to control LEDs or switches on external devices through limited rearrangement of the pins on the board. The ability to program directionality for individual applications and the functionality to handle interrupts and analog signals make application code developed for one purpose easily extendable to other applications

Q. What r process states?

What is a Process?

A process is a program in execution. Process is not as same as program code but a lot more than it. A process is an 'active' entity as opposed to program which is considered to be a 'passive' entity. Attributes held by process include hardware state, memory, CPU etc.

Process memory is divided into four sections for efficient working :

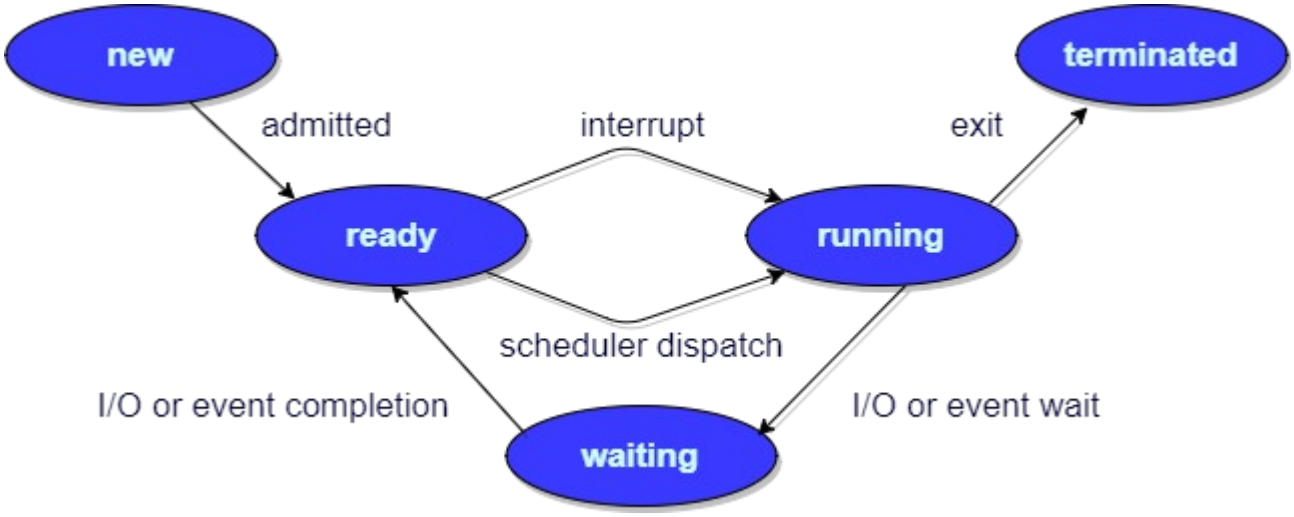
- The text section is made up of the compiled program code, read in from non-volatile storage when the program is launched.
- The data section is made up the global and static variables, allocated and initialized prior to executing the main.
- The heap is used for the dynamic memory allocation, and is managed via calls to new, delete, malloc, free, etc.
- The stack is used for local variables. Space on the stack is reserved for local variables when they are declared.

PROCESS STATE

Processes can be any of the following states :

- **New** - The process is being created.
- **Ready** - The process is waiting to be assigned to a processor.
- **Running** - Instructions are being executed.
- **Waiting** - The process is waiting for some event to occur(such as an I/O completion or reception of a signal).

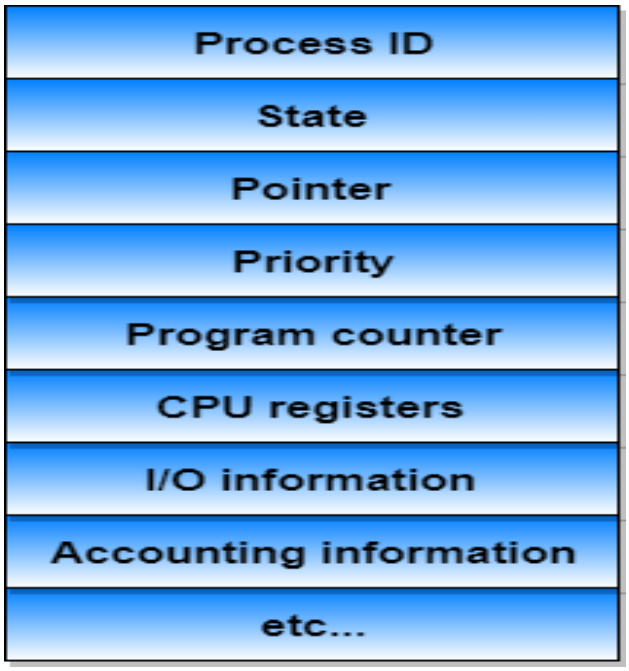
- **Terminated** - The process has finished execution.



PROCESS CONTROL BLOCK

There is a Process Control Block for each process, enclosing all the information about the process. It is a data structure, which contains the following :

- Process State - It can be running, waiting etc.
- Process ID and parent process ID.
- CPU registers and Program Counter. **Program Counter** holds the address of the next instruction to be executed for that process.
- CPU Scheduling information - Such as priority information and pointers to scheduling queues.
- Memory Management information - Eg. page tables or segment tables.
- Accounting information - user and kernel CPU time consumed, account numbers, limits, etc.
- I/O Status information - Devices allocated, open file tables, etc.



Process and thread Diff.

Mutex semaphore Diff.

What is shared memory and message queue? About Uart protocol.

Diff betw Tcp & udp?

What is meant by connection less and connection oriented .

Diff betw polling and interrupt give one Example of polling.

Soft and hard rtos Diff.

Monolithic and micro kernel Diff.

What are Initrfs, initrd,Ramfs.

what is inline function? Main purpose of inline fn.