**Core dump:**

Core dump/Segmentation fault is a specific kind of error caused by accessing memory that “does not belong to you”

* When a piece of code tries to do read and write operation in a read only location in memory or freed block of memory, it is known as core dump.
* It is an error indicating memory corruption.

Common segmentation fault scenarios:

* Modifying a string literal:

The below program may crash (gives segmentation fault error) because the line

\*(str+1) = n tries to write a read only memory.

Int main()

{

Char \*str;

/\* Stored in read only part of data segment \*/

Str=”GfG”;

/\* Problem trying to modify read only memory \*/

\*(str+1)=’n’;

return 0;

}

Output: Abnormal termination of program.

Accessing an address that is freed:

Here in the below code, the pointer p is dereferenced after freeing the memory block,which is not allowed by the compiler. So it produce the error segment fault or abnormal program termination at run time.

#include<stdio.h>

#include<alloc.h>

Int main()

{

//allocating memory to p

Int \*p= malloc(8);

\*p=100;

//deallocating the space allocated to p

free(p);

//core dump/segmentation fault

// as now this statement is illegal

\*p=110;

return 0;

**output:** Abnormal termination of program.

**Storage Classes:**

Based on the storage location,initial value,scope and lifetime the storage classes are divided into 4 types. One is auto/local variables,register, static and global storage classes.

Auto/local variables will be stored at stack memory location and the initial value will be garbage value and the scope and lifetime will be within the block or within the function where we are declaring.

Register storage class will be stored at cpu register with the default value garbage value and the scope, lifetime will be within the function or block where we are declaring.

Static storage class it will be stored at data segment based in the initialised or uninitialsed, here in data segment it will be again divided into 2 types—initialized and uninitialized and the default value is 0. Scope is within the function if it a local static or if it a global static within the file we can’t extern the static variable to out of the file. And coming to lifetime it will persist between the function calls means throughout the programme.

Extern storage class they will be stored in the data segment based on initialized or uninitialized. The default value is 0 and the lifetime and scope is throughout the programme.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Storage specifier | Storage | Initial value | Scope | Life |
| Auto | Stack | Garbage | Within block | End of block |
| Extern | Data segment | Zero | Global multiple files | Till end of program |
| Static | Data segment | Zero | Within block | Till end of program |
| Register | CPU register | Garbage | Within block | End of block |

**Structure Padding:**

Adding more bytes to the structure to implement or Speedup the CPU Process.

Let us take one example, structure which is having 3 elements char,int,double, the actual size of the structure is 9bytes but due to structure padding it will become 12 bytes.

Initially the compiler will assign 1 byte to the char variable but the remaining 3 bytes will not be sufficient for the next element so it will add 3 bytes padding.

So the compiler will not assign 2 segments for one element.

We can avoid structure padding by using pragma packing.

#include<stdio.h>

//to force compiler to use 1-byte packaging

#pragma pack(1)

Struct s

{

int i;

char ch;

double d;

};

Int main()

{

struct s A;

printf(“Size of A is %d”,sizeof(A));

}

**Output**: Sizeof A is: 13

**BIT FIELD:**

Variables that are defined using a predefined width or size are called bit fields. This bit field can leave more than a single bit. The format and syntax of bit-filed declaration inside a structure is something like this.

Struct

{

Data-type[nameofmember]:width\_of\_Bit-filed;

};

Data-type: defines the data type which establishes how a bit-field’s value will be represented and interpreted. The data type can be of simple integer, signed integer, or unsigned integer.

Nameofmember: defines the name of the bit-field member within the structure.

Width\_of\_bitfield: specifies the number of bits required in the bit-filed. It is to be noted that the width of the bit filed should have to be lesser than or at least equal to the width of the specified type.

#include<stdio.h>

//a simple representation of the date

Struct date {

Unsigned int d;

Unsigned int m;

Unsigned int y;

};

Int main()

{

Printf(“Size of date is %lu bytes\n”,sizeof(struct date));

Struct date dt= {31,12,2014};

Printf(“Date is %d/%d/%d”, dt.d,dt.m,dt.y);

}

Output: Size of date is 12 bytes

Date is 31/12/2014.

DHCP:

We are using the DHCP protocol to allocate the ip address dynamically. Whenever we are coming to a network we need a ip address to access that network.

For example, if you are in home or college or office there internally we are connecting to the wifi routers where router act as server and mobile/laptop act as client.

Whenever client wants an ip address it will send a DHCP(discover) broadcast packet, it will send to all the services available that network, if server is configured with DHCP in it they will respond with offer packets. I am trying to give this ip whether you are interested or not then client will identify that packet if he is interested it will respond with reply packet.

Before that it will check that there is any ip conflict in the same network for the other client if it is not there it will send a reply packet. Then you will get a final acknowledgement to the client so that it will get ip address with some lease time.

Client side packets will always be the broadcast packets and coming the server side if that broadcast flag is true then it is always broadcast if it is set as false then it will be the unicast reply.

Renew and release packets also unicast packets. Rebind will be the broadcast packet.

**DEAD LOCK:**

It is a situation when the process or thread acquired the resource for the long time not releasing and waiting for other resources. And it will happen in the continue like infinite loop.

If you see one example, process 1 acquired the resource 1 it and process 2 acquired resource 2 ,p1 wants to acquire r2 to complete the task and p2 need to acquire r2 to complete the task at that time the both processes don’t want to release the resources that time deadlock will happen.

In a situation or a condition (problem) it is blocking something, when 2 or more processes sharing the same resource or effectively preventing each other from accessing the resources that you called as a deadlock.

For example, p1 is holding the r1 resource while p2 holding the r2 resources. Whereas p2 requesing to access r1 and p1 requesting to access r2.

Until p1 releases the r1,p2 cannot access the r1 in the same way until p2 releases r2, p1 cannot access the r2.

One process is requesting to other resource which is already allocated to other process.

If both the processes are in waiting state that state, you call it as a deadlock state.

Conditions for deadlock:

1.Mutual explosion: (resource cannot sharable) only one process can use a resource at a time (the process can enter into critical session and execute the task,it has to follow the sequence)

2.process are in the hold and wait state: ( A process is holding one resource and waiting for another)

3. no pre-emption: (Voluntarily one process acquire by the resource can release by itself)

One process is already holding the resource and want to connect to the other resource but it is not willing to release the resource which acquired by itself.

4. Circular wait: let us take the 4 processes A,B,C,D

* A acquired some resources which is need by B
* B acquired some resources which is need by C
* C acquired some resources which is need by D
* D acquired some resources which is need by A

**OSPF BUGS:**

* “Show ospf statistics” – displaying empty tables when ospf process is not running., put on if condition to check whether process is up or not.
* Lsdb sequence no is unsigned int as per RRC but it is signed int in code.
* When ospf session removed and added (resumed), sequence number is reset at sender, handled case accordingly on receiving end.

**OSPF:**

**Routing:**  routing is a process which is performed by layer3 (or network layer) devices in order to deliver the packet by choosing an optimal path from one network to another.

**Static routing:** in this process the routing is built by the administrator manually.

**Default routing:** in this method the routing is done to single router called default router. It doesn’t matter to which the packet is belong to but the packet is forwarded to default router. The default router is configured by administrator itself.

**Dynamic routing:**  in this method the router itself built the routing table dynamically based on the current link state, this process uses the protocol like OSPF and RIP to discover the routes in the network.

Routing table fields: destination network, net mask, gateway, interface and metric.

**OSPF stands for Open shortest path first which is a standard link state protocol,** designed to scale the route effectively using shortest path first algorithm.

The key features of OSPF are:

* OSPF is a classless protocol
* OSPF employs a hierarchical network design using Areas.
* OSPF will form neighbour relationships with adjacent routes in the same area.
* Instead of advertising the distance to connected networks, OSPF advertises the status of directly connected links using Link-State Advertisements (LSAs).
* OSPF sends updates (LSAs) when there is a change to one of its links and will only send the change in the update. LSAs are additionally refreshed every 3o minutes.
* OSPF traffic is multicast either to address 224.0.0.5 (all OSPF routers) or 224.0.0.6 (all Designated Routers).
* OSPF supports only IP routing.
* OSPF routes have an administrative distance is 110.
* OSPF uses cost as its metric, which is computed based on the bandwidth of the link. OSPF has no hop-count limit.

OSPF finds the shortest path from source to destination based on OSPF database which is identical to the entire router in an area. OSPF database is formed on neighbour ship table, routing table and topological table.

OSPF routers will only become neighbours to adjacent router only if the following parameters are satisfied.

* A Hello packet are identical on each other.
* Same area ID and unique router id.
* Same area type (stub,NSSA,etc)
* Same prefix
* Same subnet mask
* Hello interval and dead interval
* Network type (broadcast ,point-to-point ,etc)
* Same authentication.

**OSPF VS RIP DIFF:**

|  |  |
| --- | --- |
| RIP | OSPF |
| RIP has a limit of 15 hops. A RIP network that spans more than 15 hops(routers) is considered unreachable. | With OSPF, there is no limitation on the hop count |
| RIP cannot handle variable length subnet masks (VLSM). Given the shortage of IP addresses and the flexibility VLSM gives in the efficient assignment of IP addresses, this is considered a major flaw. | The intelligent use of VLSM is very useful in IP address allocation. |
| Periodic broadcasts of the full routing table will consume a large amount of bandwidth. This is a major problem with large networks especially on slow links and WAN clouds. | OSPF uses IP multicast to send link-state updates. This ensures less processing on routers that are not listening to OSPF packets. Also updates are only sent in case routing changes occur instead of periodically. This ensures a better use of bandwidth. |
| RIP converges slower than OSPF. In large networks convergence gets to be in the order of minutes. RIP routers will go through a period of a hold-down and garbage collection and will slowly time-out information that has not been received recently. This is inappropriate in large environments and cloud cause routing inconsistencies. | OSPF has better convergence than RIP. This is because routing changes are propagated instantaneously and not periodically. |
| RIP has no concept of network delays and link costs. Routing decisions are based on hop counts. The path with the lowest hop count to the destination is always preferred even if the longer path has a better aggregate link bandwidth and slower delays. | OSPF allows for better load balancing based on the actual cost of the link. Link delays are a major factor in deciding where to send routing updates. |
| RIP networks are flat networks. There is no concept of areas or boundaries. With the introduction of classless routing and the intelligent use of aggregation and summarization, RIP networks seem to have fallen behind. | OSPF allows for a logical definition of networks where can be divided into areas. This will limit the explosion of link state updates over the whole network. This also provides a mechanism for aggregating routes and cutting down on the unnecessary propagation of subnet information. |
|  |  |

**Congession:**

when we have data rate on the channel or something based on the MTU it will take care about the how the data will be traversed.

When it will happen: it will happen in the TCP segment, when you are sending that

Tcp segment the congession will happen to avoid this we use windowing techniques. Sliding window protocol.

Real time scenario: For example, when you are sending a data with some window size, sender is sending the data with the window size is 10 coming to the receiver side it will not able to handling that many window segments at a time it need to handle the congession control.

Core packet greater than the window size the congession will happen.

**Flooding:**

Switch will do flooding on L2. It will do flooding where it doesn’t know the destination is seats and on which port it seats.

If it doing flooding it won’t change the source max or destination mac but it will send to all the other potrs in the switch. In order to know that particular mac where it seats.

If it is a VLAN it will flood in that particular VLAN only(whatever the ports existed in that VLAN).

**Ethernet Frame (Header):**

Actually we receive the packet with meta data where we have a port information and vlan information on which it receiving then after the meta data we will have a Ethernet frame where will have a destination mac, source mac and type field then we have payload upper layer payload information.

How you decide receiving packet is a ip packet or any other protocol packet?

Type field if it is 0x0800 then it is a ip packet the upper layer information is ip packet or if it is a 802.1Q then it is having as a tagged frame.

Ethernet frame is a L2 frame, it will have a L2 information based on the source mac, destination mac and also it will have type or length field to decide what type of upper layer protocol it is or if it is only length so it will add the length information apart from that it will have a payload and CRC (cyclic redundancy check) for error control, it is in the L2 level.

And coming to L1 level it will have a preamble and start of frame delimiter and interframe gap.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| PREAMBLE –7BYTES | SFD – 1BYTE | DESTINATION ADDRESS—6BYTES | SOURCE ADDRESS-6 BYTES | LENGTH—2 BYTES | DATA-46 TO 1500 BYTES | CRC-4BYTES |

**Switch functions at layer 2:**

Switch is a layer 2 device which works on the basis of MAC address of a device. Switch mainly performs these functions.

1. Learning: the switch learns the MAC address of the device on the switch porton which it receives the frame.
2. Forwarding: the switch does 2 types of message forwarding.
3. Unicast: the switch unicast the frame to the destination only when it has an entry for destination MAC address table.
4. Unknown Unicast: when a switch receives a unicast frame for a destination for which switch has no entry in its MAC table then the switch simply broadcast the frame through all ports. This is knows as flooding.
5. Filtering: the frame will be forwarded through that switch port only for which the switch has already learned the MAC address in its MAC table.
6. Loop avoidance: For redundancy, two switches are connected to each other through two links which can also result in layer 2 loops. These loops are avoided by switch by using STP (spanning tree protocol).

NOTE: An entry in the switch MAC table also known as CAM (content addressable memory), can remain upto for 300 seconds. When a frame is received for a destination MAC address, the time limit of 300 seconds get reset. MAC table has 4 entries.

* Port number: The switch port attached to the destination MAC.
* MAC address: MAC address of that host which is attached to that switch port.
* Type: It tells us about how the switch has learned the MAC address of the host i.e, static or dynamic. If the entry is added manually then it will be static otherwise it will be dynamic.
* VLAN: It tells about to which VLAN the host , attached to that switch port, belongs to.

How switch learns the MAC address:

The switch will update it’s MAC table only when it receives any frame from the host

**Memory leak**:

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

#include<stio.h>

Void f()

{

Int \*ptr = (int \*)malloc(sizeof (int));

Return; // return without freeing ptr

}

**Local static variables:**

A local static variable is a variable whose lifetime doesn’t end with a function call where it is declared. It extends for the lifetime of complete program. Static variables can be used to count the number of times a function is called. Static variables get the default value is ‘0’.

#incldue<stdio.h>

Void fun()

{

Static int x; // static variables get the default value as ‘0’

Printf(“%d”,x);

X=x+1;

}

Int main()

{

Fun();

Fun();

Return 0;

}

o/p: 0 1

**static functions**:

in c, functions are global by default. The “static” keyword before a function name makes it static. Unlike global functions in c, access to static functions is restricted to the file where they are declared. Therefore, when we want to restrict access to functions, we make them static. Another reason for making functions static can be reuse of the same function name in other files.

**TCP and UDP differences**:

|  |  |
| --- | --- |
| TCP | UDP |
| Connection made before application messages are exchanged. | Connection not made before application messages are exchanged. |
| For applications needing more reliability and less speed. Ex: File transfer, Email | For applications needing more speedy and less reliability. Ex: multimedia. |
| Messages will be delivered in order and without errors | No guarantee that the messages will be delivered in order and with out errors. |
| Data segments rearranged in required order | All segments are independent, therefore has no inherent order specification. |
| ACK is received | ACK is not received |
| Has the congestion control mechanism | No flow control option |
| Resends erroneous segments | Discards erroneous segment. |

**Volatile:**

The volatile qualifier is applied to a variable when we declare it, it is used to tell the compiler that the value may change at any time.

The volatile keyword cannot remove the memory management.

It cannot cache the variables in register

The value cannot change in order of assignment.

The volatile keyword is intended to prevent the compiler from applying any optimizations on objects that can change in ways that cannot be determined by the compiler.

Objects declared as volatile are omitted from optimization because their values can be changed by code outside the scope of current code at any time.

**Heap allocation and Stack allocation**:

**Stack allocation**:

The allocation happens on contiguous blocks of memory. We call it a stack memory allocation because the allocation happens in the function call stack. The size of memory to be allocated is known to the compiler and whenever a function is called, its variables get memory allocated on the stack. And whenever the function call is over, the memory for the variables is deallocated. This all happens using some predefined routines in the compiler. A programmer does not have to worry about memory allocation and deallocation of stack variables, also called as temporary memory allocation because as soon as the method finishes its execution all the data belongs to that method flushes out from the stack automatically.

**Heap allocation**:

The memory is allocated during the execution of instructions written by programmers. Note that the name heap has nothing to do with the heap data structure. It is called heap because it is a pile of memory space available to programmers to allocate and deallocate. Every time when we made an object it always creates in heap space and the referencing information to these objects are always stored in stack memory. Heap memory allocation is not as safe as stack memory allocation was because the data stored in this space is accessible or visible to all threads. If a programmer doesnot handle this memory well, a memory leak can happen in the program.

|  |  |  |
| --- | --- | --- |
| Parameter | Stack | Heap |
| Basic | Memory is allocated in a contiguous block | Memory is allocated in any random order |
| Allocation and deallocation | Automatic by compiler instructions | Manual by programmer |
| Cost | Less | More |
| Implementation | Easy | Hard |
| Access time | Faster | Slower |
| Main iisue | Shortage of memory | Memory fragmentation |
| Flexibility | Fixed-size | Resizing is possible |
| Data type structure | Linear | Hierarchical |

**Structure and Union diff**:

|  |  |  |
| --- | --- | --- |
|  | Structure | Union |
| Keyword | “struct” is used to define structure | “union” is used to define union |
| Size | When a variable is associated with a structure, the compiler allocates the memory for each member. The sizeof structure is greater than or equal to the sum of sizes of its members | When a variable is associated with a union, the compiler allocates the memory by considering size of the largest memory. So, size of union is equal to the size of largest member. |
| Memory | Each member within a structure is assigned unique storage area of location. | Memory allocated is shared by individual members of union. |
| Value altering | Altering the value of a member will not affect other members of structure | Altering the value of any of the member will alter other member values. |
| Accessing members | Individual member can be accessed at a time | Only one member can be accessed at a time |
| Initialization of members | Several members of a structure can initiate at once | Only the first member of a union can be initiated. |