Date 16/10/18
Page

UNIX Operating System

OS:

It is the interface between the hardware and the applications.

Kernel is the core part of OS.

It has 3 major parts

1. process
2. Task
3. Thread

1. process:

process is an instance of a program that is being executed

2. Thread:

A Thread is a light weighted process which intern Consists of program Counter, Stackpointer and a Set of registers.

3 Task:

A Task is a Set of instructions that is being executed

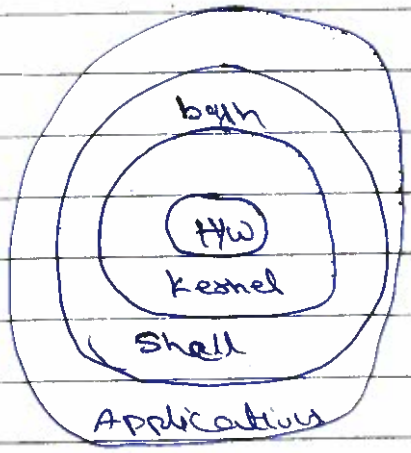


fig: OS

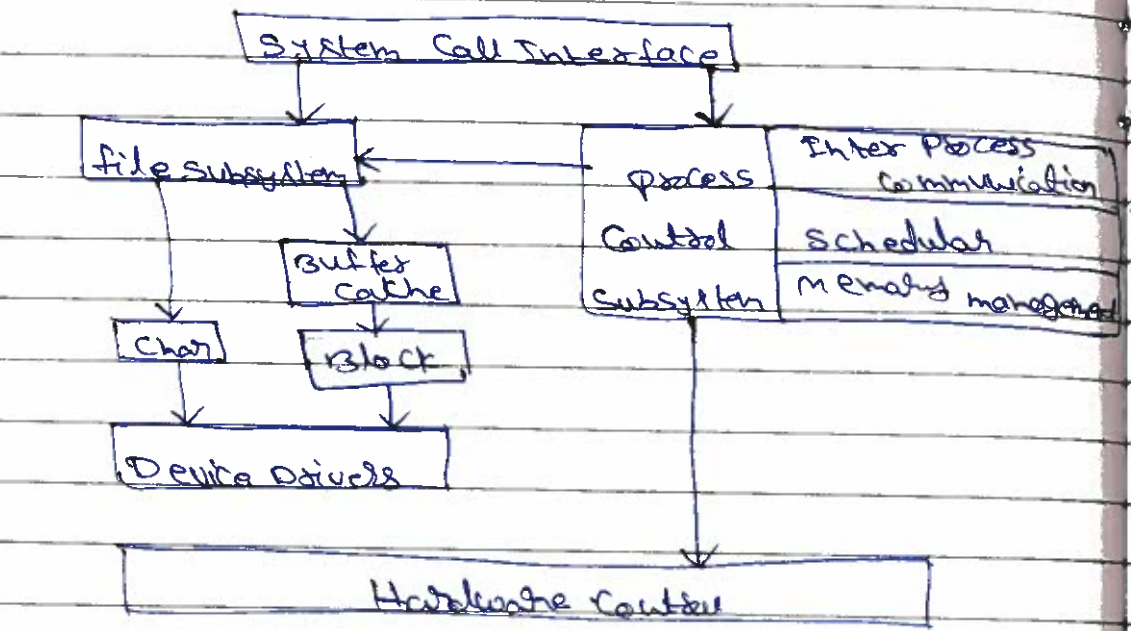


fig: UNIX operating system

System Call Interface is the interface b/w the applications and the kernal. It is used to take the input from the application and give it to the file or process subsystem to process it.

file subsystem handles character & Block Drivers character driver is used to send data in form of character by character. It will be directly connected to the file subsystem.

Block Device driver is used to send the data in blocks.

→ keyboard is the example for character device driver

→ Example for block device driver is CD-ROM, Hard disk

Process control subsystem handles inter process communication, scheduler & memory management

Inter process Communication:-

The communication between more than 1 process is nothing but inter process Communication. This can be achived by using the following IPC methods

1. pipes
2. message queue
3. Semaforce
4. Socate programming
5. Shared memory

Scheduler:-

scheduler is used to schedule the process with Synchronization Synchronization

Ex: If P_1, P_2, P_3 are the process the scheduler will execute all this process with different methods.

1. first in first out (fIfo)
2. Round Robin Scheduling
3. Priority based Scheduling

memory management:

memory management is used to utilise the memory in a specific way.

UNIX OS:-

UNIX is a multitasking, multiprogramming and multi scheduling type of OS.

OS deals with 3 major parts.

1. Process management
2. file management
3. Memory management

UNIX/ LINUX

1.) login	(7) head	(13) Shutdown
2) cd	(8) tail	(14) ls
3) mkdir	(9) grep	(15) ls -a
4) Rmdir	(10) pwd	(16) echo
5) CP	(11) Passwd	
6) mv	(12) clear	

1.) login:-

It is used to login into the system using username & Password

2) CD:-

CD is used to move from one directory to other directory or from the present working directory to home directory.

3) Mkdir:-

It is used to create the directory

4) Rmdir:-

It is used to remove the existing directory.

5) CP:-

CP is used to copy one file to other file.

Date
Page

6) mv is used move file from one place to other place

7) head:-
Head is used to print the first few lines in a file

8) tail:-
It is used to print last two lines in a file

9) grep:-
Grep is used to find a particular word in a file or directory.

10) Pwd:-
Pwd is used to get present working directory

11) Passwd:-
It is used to create password and open the Password

12) clear:-
It is used to clear the screen

13) Shutdown:-
It is used to shutdown the PC using OS.

14) ls:-
ls is used to list all the files in a directory from the given path

19.18-a:-

It is used to list all the files including hidden files in a directory.

16) echo:-

It is same as Printf. It is used to print.

18/10/19

Process management

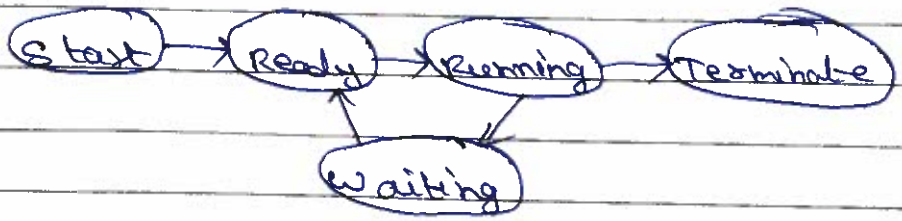
Process management alings the process in a particular manner & executes them one after one based on the scheduling mechanism.

The schedular schedules the process for the CPU in such a way which one executes first & which one follows it

The process basically has the below mentioned stages

1. start/ create
2. Ready
3. Running
4. exit
5. Terminate/exit

Process states



Date
Page

1. preemptive Scheduling
2. Non preemptive scheduling

In preemptive scheduling the low priority task which is executing will go to wait state when a high priority task preempts the low priority task.

In non preemptive scheduling the low priority task continues executing even if the high priority task occurs.

Types of Schedeling:-

1. FCFS
2. Shortest job first (SJF)
3. Round Robin scheduling
4. priority based scheduling

→ In FCFS the process which comes first will be served first.

→ In SJF the process which is having shortest time frame will be served first.

→ In Round Robin scheduling each & every process will get equal time & it will execute one after the other by time sharing process.

→ In priority based scheduling the process which is having highest priority will get the CPU

Aging:

Aging is a technique that gradually increments the priority of a low priority task to get CPU time for it.

Priority based scheduling

Process	Arrival time	Burst time	Priority
P1	0	5	2
P2	2	3	3
P3	5	8	4
P4	7	10	1

process	Arrival	Burst	Priority
P1	0	5	2
P2	5	4	3
P3	3	8	4
P4	7	10	1

P1	0	5	2
P2	5	3	3
P4	7	10	1
P2	17	1	3
P3	18	8	4

⇒ Input the process based on the priority, burst time and Arrival time.

⇒ Apply FIFO scheduling algorithm.

Date 22/10/18
Page

① Priority Inversion
② Priority Inheritance

Priority Inversion defines a problem where as priority Inheritance provides a solution for it.

Priority Inversion is an problem where the low priority task may not get the CPU time for execution.

Priority Inheritance is an solution for the problem defined by priority inversion.

In priority inheritance the process which will not get the CPU time gets its priority incremented for each and every process execution time, so, that the low priority task also gets the CPU at some part of time.

Memory management:

It is a process of allocating deallocating, keep track of memory usage (which part of memory is used & which part of the memory is unused), making access faster.

→ main memory is the virtual memory it will take the instructions from secondary memory and executes

→ Secondary memory is the physical memory

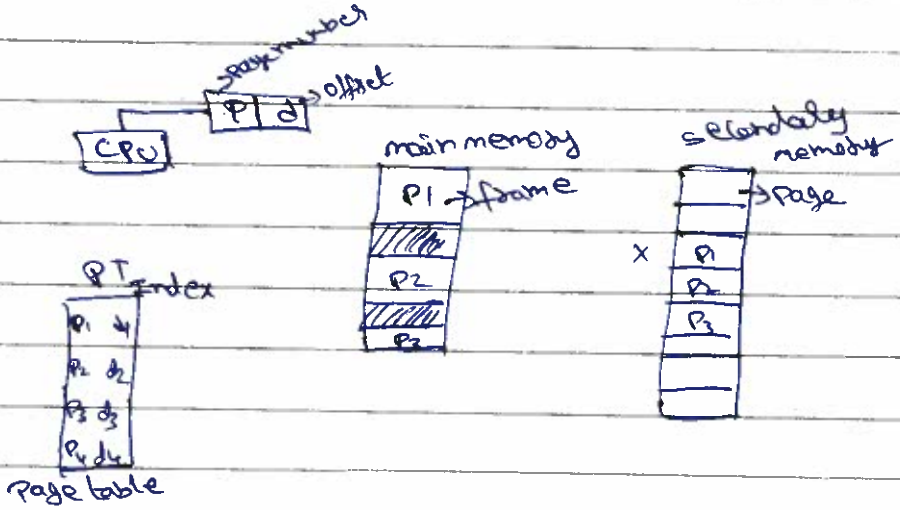
which Consists of Code and instructions.

→ Example for main memory is RAM, It will be 4, 8 Kbs in micro controllers and for laptops is 4 GB.

→ Hard disk, (or) Drives are the example for Secondary memory.

23/10/18 ** Paging:-

Paging is a ~~the~~ technique where the main memory and the Secondary memory is divided into equal size of pages (or) blocks and the memory will be allocated in Contigious ~~and~~ (or) non-Contigious fashion.



In Secondary memory the memory is divided into equal size of blocks called pages and in main memory the memory is divided into same size of blocks and with respective to Secondary memory which is called as frames.

In main memory the process can be arranged in Contigious (or) non-Contigious fashion.

if the first process is accessed the second, third and soo on can accessed using first process. It will be arranged in form of data structure such that P_1 knows P_2, P_2 knows P_3 and soo on.

The P_1 process also can be divided into some equal blocks and can arrange that in non contigious fashion in main memory. The first, second, third blocks can be find out by the CPU using Page tables.

Swapping:

Swapping is a process of allocating one individual block for each and every process. The process which is used very minimal can be swapped out and the process which is having high priority can be swapped in. Using swapping

Advantages of swapping:

→ Swapping is very faster as it deals with process to process.

Disadvantages:

memory wastage will be there due to the memory allocated for the complete process

File Management

file management is used to manage and access the files from the secondary devices to the main memory. it consists of

1. Directories
2. files
3. special files

mk dir xyz → It is used create a directory.
rm dir xyz → It is used to remove the directory
vi → It is used to create a file
cp → It is used to copy the file
mv → is used to move the file.
cat → is used to print the content of a file.
cd → is used to move to a particular directory
cd.. → is used to come back to home directory
pwd → It provides the current working directory
head → It is used to print the first few lines of a file
Tail → is used to print the last few lines of a file
ls → is used to list out the files present in current working directory
ls -a → is used to list out all the files including hidden files in current working directory

Date
Page

Interprocess Communication

Exchanging information b/w one with the other process is called as interprocess Communication it can be achived using

1. Pipes
2. message queues
3. Shared memory
4. Socket Programming
5. Semaphores.

fork:-

fork is used to create child process. The child process will be having address zero. where as parent process will be having non zero value as address.

Pipes:-

There are 2 types of pipes in OS

i. unnamed pipes

unnamed pipes is used to communicate b/w known process

Ex:- Parent and Child.

ii. named pipes:-

named pipes is used to communicate b/w two unknown process.

Unnamed pipes:-

open(int pid, int access) — pid: process ID; access: RD-only, WR-only, RW

read(int fd, char *buffer, sizeof(buffer))

write(int fd, char *buffer, sizeof(buffer))

close(int pid)

```
#include<stdio.h>
#define write 1
#define read 0
Void main()
{
   Char a[20];
   int  fd[2];
    pipe(fd);
if(fork()==0) {     //child
  close(fd[0]);
 write(fd[1], "Hello Parent\n", 15);
 close(fd[1]);
}
else    //Parent
 {
  close(fd[1]);
  read(fd[0], a, 15);
  Printf("%s", a);
 close(fd[0]);
}}
```

24/10/18 **Named pipes:-**

Process #1

```
#include<stdio.h>
#include<sys/start.h>
#include<fcntl.h>
#define buffer "HelloWorld"
main() {
 int fd1;                                  fcntl.h
 fd1=open("mypipe", "O-WRONLY");
 write(fd1, buffer, sizeof(buffer));
} close(fd1);
```

Process#2

```
#include <stdio.h>
#include <sys/stat.h>
#include <fcntl.h>
 Char  data [100];
 main() {
  int  fd2;
fd2 = open (" mypipe", O_RDONLY);
      read (fd2, data, 10);
       printf (" %s", data);
       close (fd2);
    }
```

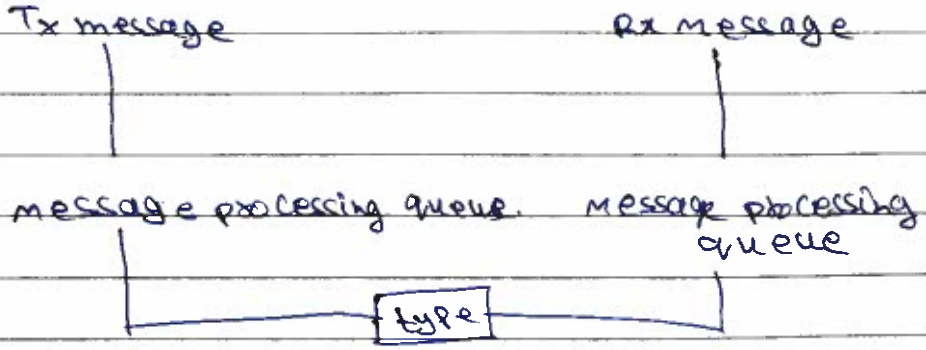
mknod mypipe p → creating a pipe

mknod ("mypipe", S_IFIFO | 0, 0) → nowait for SystemCall

└ sys/stat.h)

Message Queues

Message queues are used for interprocess Communication that transfers data from one process to other process using the type of a message



```
struct msg-queue
{
    long type;
    char text[100];
};
int msgget(key_t key, int permission);

int msgsnd(int msgid, struct msg-queue *msg-queue,
           sizeof(msg-queue), int flag)
                                  ↳ IPC_NOWAIT

int msgrcv(int msgid, struct msg-queue *msg-queue,
           sizeof(msg-queue), long type, int flag)

int msgctl(int msgid, int permission, int flag)
                                  ↳ IPC_RMID
```

Ex. (message queues)

```
#include <stdio.h>
#include <sys/msg.h>
#include <sys/ipc.h>
#include <sys/stat.h>
#include <sys/types.h>

void main(){ ——— struct msg-que{
  long type;
  char text[100];
};
struct msg-que msg-queue;
msg-queue.type = 4;
msg-queue.text = "helloworld";
key_t key = 100; → sys/types.h
```

Date
Page

```
int msgid = msgget(key, IPC_CREAT|0666);
msgsnd(msgid, &msg_queue, sizeof(msg_queue, 0);
// msgctl(msgid, IPC_RMID, 0);
}.
```

delete any message queue

Rx (messagequeue)

```
#include <stdio.h>
#include <sys/stat.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <sys/types.h>
void main() {
    Struct msg_queue {
        long type;
        char text[100];
    };
    Struct msg_queue msg_queue;
    //msg_queue.type=1;
    key_t key=100;
int msgid = msgget(key, IPC_CREAT|0666);
msgrcv(msgid, &msg_queue, sizeof(msg_queue), 1, 0);
// msgctl(msgid, IPC_RMID, 0);
}.
```

26/10/18 **Shared memory:**

Shared memory is a process of utilizing memory by all the process with synchronization.

```
int Shmget(key_t key, size_t size, int permission)
void *shmat(int shmid, (void *)addr, int flag)
void *shmdt((void *)addr)
int shmctl(int shmid, IPC_RMID, int flag)
```

→ 'shmget' is used to create an id for the shared memory with key, size and permission. size determine the size i.e we are allocation the memory.

→ 'Shmat' is used to attach Process to the memory. It returns starting address of the memory which got allocated. It has the parameters 'Shmid', address of the Sharedmemory and flag. The address can be defined null so that it will fetch the default memory address.

→ 'Shmdt' is used to dettach the memory to the process.

Note:-

with dettach system call the shared memory cannot be destroyed.

→ 'Shmctl' is used to destroy the shared memory. The second parameter IPC_RMID is used to destroy the created shared memory.

process #1

```
#include <stdio.h>
#include <sys/shm.h>
#include <sys/ipc.h>
#include <sys/type.h>
#include <sys/stat.h>
    void main(){
        key_t key = 1234;
        size_t size = 20;
```

Date
Page

```
int Shmid = shmget(key, size, IPC_CREAT|0666);
int *ptr = (int *)shmat(Shmid, NULL, 0);
  for(int i=0; i<5; i++)
        *ptr = i;
         ptr ++;
  }
 shmdt(ptr);
 shmctl(shmid, IPC_RMID, 0);
}
```

Process #2

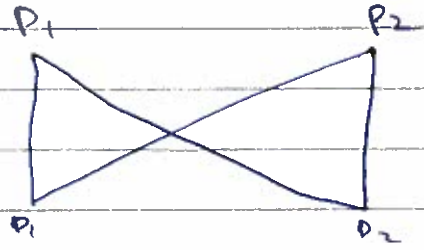
```
#include<stdio.h>
#include<sys/ipc.h>
#include<sys/shm.h>
#include<sys/type.h>
        void main(){
          key_t key = 1234;
        size_t size = 20;
int Shmid = shmget(key, size, IPC_CREAT|0666);
int *ptr = (int *) shmat(shmid, NULL, 0);
 for(int i=0; i<5; i++) {
     printf("%d", *ptr);
      ptr ++;
} shmdt(ptr);
  shmctl(shmid, IPC_RMID, 0);
}
```

Deadlock:-

If process1 is holding resource R_1 and it is waiting for resource R_2 to be free to leave resource

If process2 is holding resource R_2 and it is waiting for resource R_1 to get

release to leave R_2 and to hold R_1. which will not happen. This condition is called deadlock condition



→ ~~This~~ The issue can be overcome by using ~~Semaphores~~ Semaphores.

Eg:- The best example for Semaphores is railway signales.

These are 3 types of Semaphores

1. Binary Semaphore
2. Counting Semaphore
3. Mutex

1. Binary Semaphore:-

In this the process will be having identical key to use the resource. The process which is having highest priority can use the resource.

2. Counting semaphore:-

In Counting Semaphore there will be more than one resource and all the process with identical keys can be use those resources. All resources will be having similar lock. ~~So the~~ The Count will get incremented when the resources are get free. The Count will be zero when all the resources got occupied by the

Date
Page

process.

mutex:-

In mutex there will be only one key to utilizie the resource. The process which is holding the key can only utilize the resource. The resource cannot be used by any other high priority process using till it get unlocked by the process using it.

26/10/21

Create semaphore

semget (key_t key, int sem_num, int flag)

switch on/off semaphore

semctl (int semid, int sem_num, SET_VAL, int value on
↓
Permission

Semaphore operation (returns '0' when semaphore is locked
-1 when semaphores is not successfully locked).

semop (int semid, struct sembuf *SOP, int nsems);

struct sembuf {
unsigned short sem_num;
short sem_op;
short sem_flag;
}

Ex:

#include<ipc/sem.h>
main(){
key_t key = 2;
struct sembuf SOPS = {0, -1, 0};
sem_num, sem_op, sem_flag

```
int Semid=Semget (key, 0, IPC-CREAT|0666);
int Semctl = semCtl(Semid, 0, SETVAL, 1);
int SOP = Semop (Semid, &SOPs, 1);
if (SOP==0) {  //Semaphore locked
int Shmid = Shmget (key, 50, IPC-CREAT|0666);
int *Ptr = (int *) Shmat (shmid, NULL, 0);
    for(int i=0; i<5; i++) {
        *Ptr = i;
        Ptr.++;
}
  sleep (20);
 Shmdt (Ptr);
Semctl=Semctl(Semid, 0, SETVAL, 1);
if (Semct = =0) {
Printf (" Semaphore Set switched off ");
  }
else {
  Printf (" Semaphore still on ");
} }.
```

Rx

```
#include<ipc/sem.h>
  main() {
  key-t  key=20;
Struct Sembuf SOPS = {0, -1, 0};
int Semid = Semget (key, 0, IPC-CREAT|0666);
int Semctl: Semctl(Semid, 0, SETVAL, 1);
int SOP = Semop (Semid, &SOPs, 1);
 if (SOP = = 0) {
  int Shmid=Shmget (key, 50, IPC-CREAT|0666);
  int  *Ptr = (int *) Shmat (Shmid, NULL, 0);
   for(int  i=0;  i<5;  i++) {
```

```
printf("%d", *ptr);
    ptr++;
}
shmdt(ptr);
semctl = semctl(semid, 0, SETVAL, 1);
if (semctl == 0) {
  printf("semaphore got switched off");
  } else {
      printf("semaphores is still on");
} }
```

29/10/18

Socket Programming

Socket programming is used for interprocess communication between clients and the server.

The protocol used for wired communication is TCP/IP

The protocol used for wireless communication is UDP (User datagram protocol)

The System call for Server and client.

Server	Client
socket	socket
↓	↓
bind	Connect
↓	R/W
listen	
↓	
Accept	
↓	
R/W	

int Socket (int family, int type, int protocol)

AF-INET → will be used for internet communication

SOCKET-STREAM → used for TCP/IP communication

protocol → default protocol

int bind (int sock_id, struct sockaddr_in *sockaddr, sizeof(sockaddr))

sys/netinet/in.h.

struct sockaddr_in {
short sin_family;
u_short sin_port;
struct in_addr sin_addr;
char sin_zero[100];
}

struct in_addr {
unsigned long s_addr;
}

} predefined

int listen (int sock_id, int backlogs)

int accept (int cl_id, struct sockaddr_in *clientaddr, &clientaddr);

client

int connect (int client_id, struct sockaddr_in *client_addr) sizeof(client addr));

Date
Page

Server program

```
#include <sys/socket.h>
#include <sys/netinet/in.h>
#include <sys/netinet/ip.h>
  char a[20];
  main(){
  int client_id;
int  sock_id = socket(AF_INET, SOCK_STREM, 0);
  struct sockaddr_in sock;
     sock.sin_family = AF_INET;
     sock.sin_port = htons(8000); //(for changing littleendin to
                                    Bigendin)
int  b_in = bind(sock_id, &sock, sizeof(sock));
int  l_in = listen(sock_id, 1);
   struct sockaddr_in client;
int  client_in = accept(sock_id, &client, &sizeof(client));
     read(client_id, a, sizeof(a));
     printf("%s", a);
}
```

Client program

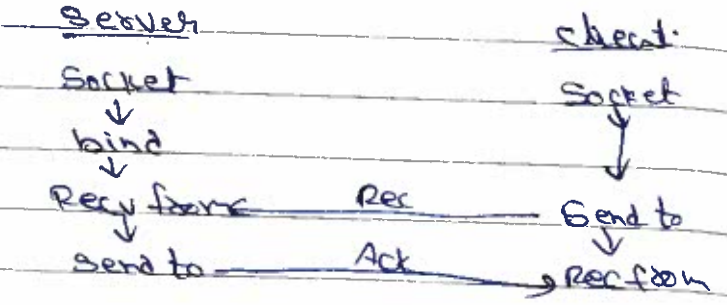
```
# include <sys/socket.h>
#include <sys/netinet/in.h>
#include <sys/netinet/ip.h>
 int main() {
     int socket_id;
 socket_id = socket(AF_INET, SOCK_STREM, 0);
  struct sockaddr_in client;
  client.sin_family = AF_INET;
  client.sin_port = htons(8000);
  client.sin_addr.s_addr = inet("10.156.15.1");
int  conn_id = connect(client_id, &client, sizeof(client));
}    write(conn_id, "hello world", 20);
```

30/10/18

UDP (User datagram protocol)

Difference b/w TCP/IP & UDP

TCP/IP	UDP
1) Connection oriented	1. Wireless
2) It is slower than UDP	2) faster than TCP/IP
3) There is error recovery management	3) There is no error recovery management.
4) Ack will be there in TCP/IP	4) Ack is not available.



int socket (int family, int type, protocol)

family	type	
AF_INET - IPV4	SOCK_STREAM (TCP/IP)	0
AF_INETG - IPV6	SOCK_DGRAM (UDP)	0

int bind (int sockid, struct sockaddr_in *sockaddr, sizeof(sockaddr));

int sendto (int sockid, char *buffer, sizeof(buffer), int flag, struct sockaddr_in. *cliaddr, sizeof(cliaddr));

int recvfrom (int sockid, const char *buffer, sizeof(buffer), int flag, struct sockaddr_in. *cliaddr, sizeof(cliaddr));

Date
Page

Server side program for UDP

```
#include<Sys/socket.h>
#include<netinet/in.h>
#include<netinet/ip.h>
   Const char buffer[100];
    main() {
int sockid= socket(AF_INET, SOCK_DGRAM, 0);
 struct sockaddr_in sockaddr, cliaddr;
     sockaddr.sin_family = AF_INET;
     Sockaddr.sin_port = htons(8080);
    sockaddr.sin_addr.s_addr = net_inet("10.156.15.1");
 memset(& Sockaddr, 0, sizeof(Sockaddr));
 memset(& cliaddr, 0, sizeof(cliaddr));

int b= bind(sockid, & sockaddr, sizeof(sockaddr));
recfrom(sockid, buffer, sizeof(buffer), 0, & cliaddr,
                & Sizeof(cliaddr));
   Printf("%s", buffer);
sendto(sockid, buffer, sizeof(buffer), 0, & cliaddr,
          sizeof(cliaddr);
   close(sockid);
}
```

Client side Program for UDP

```
#include<Sys/Socket.h>
#include<netinet/in.h>
#include<netinet/ip.h>
  Const char &buffer[100];
   main() {
int sockid = socket(AF_INET, SOCK_DGRAM, 0);
 Struct sockaddr.in, cliaddr;
 Sockaddr.sin_family = Af_INET;
 Sockaddr.sin_addr.s_addr = net_inet("10.156.11.1");
```

```
memset(&cliaddr, 0, sizeof(cliaddr));

sendto(sockid, buffer, sizeof(buffer), 0, &cliaddr,
        sizeof(cliaddr));

recvfrom(sockid, buffer, sizeof(buffer), 0, &cliaddr,
        &sizeof(cliaddr));
  printf("%s", buffer);
  close(sock_id);
}
```

Posix Threads

A Thread is a light weighted process

1. A thread can be created faster
2. Termination is easy
3. Context Switching between the threads is faster
4. Communication between threads is much faster

create a thread — attribute for synchronization. Default it is zero

```
int pthread_creat(pthread_t * thread, Pthread_attr_t *attr,
        void *fun_routine, void *arg)

int pthread_exit(pthread_t  thread id);// Terminate
long pthread_self(pthread_t threadid);
int pthread_join(pthread_t  pthread, void **retval);
```

→ pthread_creat is used to create a thread.

◆ The first parameter is the threadid, 2nd parameter ~~is used~~ attribute is the attribute used for thread synchronization default it can be zero, 3rd parameter is the function routine. The

function which needs to be during the thread creation can be passed here, 4th parameter argument is argument that is passed to the function. The ~~return~~ data type ~~of~~ should be void.

→ Pthread_exit is used for terminate a thread
→ Pthread_self provides the thread id which is under execution
→ Pthread_join will synchronize the threads. It will make the other threads to wait until the current execution thread completes its execution. first parameter is thread_id, 2nd parameter is the returnval of thread during its exit.

31/10/18

Initialization of mutex is as follows.

pthread_mutex_t mymutex PTREAD_MUTEX_INITIALIZER
pthread_mutex_lock (pthread_mutex_t *mymutex)
pthread_mutex_trylock (pthread_mutex_t *mymutex)
pthread_mutex_unlock (pthread_mutex_t *mymutex)
pthread_mutex_destroy (pthread_mutex_t *mymutex)
pthread_cond_wait (Pthread_mutex_t *mymutex,
pthread_cond_t *my Cond)
pthread_cond_signal (Pthread_~~mutex~~Cond_t *myCond)
~~pthread_mutex_t mymutex~~
pthread_cond_t myCond PTREAD_COND_INITALIZER

program for thread

```
#include <Pthread.h>
  void fun() {
    Printf ("hello");
    pthread_exit(NULL);
  }
  main() {
```

```
    pthread_t tid;
    pthread_create (&tid, NULL, fun, NULL)
    pthread_exit (NULL);
    }
```

For having more than 1 thread in a program

```
#include <pthread.h>
  void fun (void *arg) {
    char *a = (char*) arg;
    printf(" %s", a);
    pthread_exit (NULL);
   }
   main(){
    pthread_t tid1, tid2;
   pthread_create (&tid1, NULL, fun, "hello");
   pthread_create (&tid2, NULL, fun, " hai");
   pthread_join (tid1, NULL);
   pthread_join (tid2, NULL);
   pthread_exit(NULL);
  }
```

```
#include <pthread.h>
   void fun (void *arg){
  pthread_exit (arg);
  } char *st, *st1;
 main () {
    pthread_t tid1, tid2;
 pthread_create (&tid1, NULL, fun, "thread1");
 pthread_create (&tid2, NULL, fun, "thread2");
 pthread_join (&tid1, &st);
```

Date
Page

```
pthread_join(&tid2, &st1);
printf("The thread exited is %s", st);
printf("The thread exited is %s", st1);
Pthread_exit(0);
}
```

How to use mutex? Signal condition wait.

```
#include<pthread.h>
pthread_mutex_t mymutex PTREAD_MUTEX_INITALIZER
pthread_mutex_t mymutex PTREAD_MUTEX_INITALIZER;
pthread_cond_t myCond PTREAD_COND_INITALIZER
    int a=2;
main(){
  Pthread tid1, tid2;
  Pthread_create(&tid1, NULL, thread1, NULL);
  Pthread_create(&tid2, NULL, thread2, NULL);
  Pthread_join(&tid1, NULL);
  Pthread_join(&tid2, NULL);
  Pthread_exit(NULL);
}
        void thread1(void){
  Pthread_mutex_lock(&mymutex);
  Printf("I am in thread1");
  if(a!=5)
  Pthread_cond_wait(&mymutex, &myCond)
  Printf("%d", a);
  Pthread_exit(NULL)
  Pthread_mutex_unlock &mymutex);
        Void Thread2(void){
          Pthread_mutex_lock(&mymutex)
      printf("I am in thread2\n");
        a=5;
```

```
    pthread_cond_signal(&mymutex, &mycond);
    printf(" Thread2 existing \n");
    pthread_mutex_unlock(&mymutex);
}
```

→

```
#include <pthread.h>
pthread_mutex_t mymutex PTREAD_MUTEX_INITALIZER
    int a=0;
    main(){
    pthread_t tid1, tid2;
    pthread_create(&tid1, NULL, thread1, NULL);
    pthread_create(&tid2, NULL, thread2, NULL);
    pthread_join(tid1, NULL);
    pthread_join(&tid2, NULL);
    pthread_exit(NULL);
}
        void thread1(void){
    pthread_mutex_lock(&mymutex);
    for(i=0; i<65535; i++) printf("%d", pthread_self());
            a++;
    printf("%d", a);
    pthread_mutex_unlock(&mymutex);
}
        void thread2{
    pthread_mutex_lock(&mymutex);
    for(i=0; i<65535; i++) printf("%x", pthread_self());
            a--;
    printf("%d", a);
    pthread_mutex_unlock(&mymutex);
}
```

Date 5/01/18
Page

Python

Python is invented by Guido Van Rossum during early 90's and late 1985.

Python is interactive, interpreted, object oriented and beginners language.

Python is a high level language.
It is an case sensitive language.

Interactive:- python is interactive as we can interact directly with OS scripts and other programming languages

Interpreted:- python is interpreted as it takes the input of the source code and compile line by line.

object oriented:- python supports all the object oriented concepts.

Beginners language:- python is very easy and its simple for the beginner to start with python

Features:-
1. Easy to understand
2. Easy to read
3. Easy to maintain
4. Scalable
5. portable
6. GUI Programming
7. Databases
8. Interactive

Scalable:- Pythone is structured and it supports largesize programes also.

Portable:- Python supports many operating systems it is easy to move from one operating system to other and interact with many programming languages. The program written for one programming language interaction can be used for the other programming languages also.

GUI programming:- It supports Graphical user interface Programming also.

Databases:

Python supports all major commercial databases

Operators precedence in python

~, +, - → Unary operator
*, /, %, //, +, - → Arthematic operators
<<, >>, &, ^, | → Bitwise operators
<=, <, >, >= → Conditional } Relational operators.
<>, ==, != → equality }
=, +=, -=, *=, /= } Assignment operators.
is, is not → identity operator
in, not in → membership operator
AND, OR, NOT → logical operator

There are 5 datatypes in python

1. Numbers
2. Strings
3. List
4. Tuples
5. Dictionaries

Date
Page

#import → is used to include the header files or Standared libraries. python has predefined Standared libraries.

Ex: #import os

```
a = 10
b = 20.0
c = "hello"
print a
Print b
Print c
```

Strings: python string can be enclosed in single quote, double quote and (or) triple quote.

Ex: """ hello'world' """

```
a :"hello world"
Print a[2:]  // llo world
print a[2:5] // llo.
```

List: In python list will be enclosed in square brackets[]

Ex: list = [2, 'hello', 20.0].

The list can be modified

```
list[0] = 3.
list[1] = 'world'.
```

Tuples: Tuples are enclosed in parenthesis, the differences between list and tuple is

1. List is mutable
2. Tuple is immutable.

Ex: a(2, 5, 7)
print a.

Dictionaries: The Dictionaries are enclosed in curvely braces it contains keys and values.

The value will be extracted with the keys.

Ex:- a {'Vinod': 25, 'Shiva': 30, 'Lakshmana': 40}
print a['Vinod']

6/11/18

Keywords in Python

and	if	return	from
as	elif	raise	import
assert	else	true	~~with~~
class	for	false	yield → return more than one value
continue	while	global	break
in	with	nonlocal	pass
is	do	except	lambda → it a tiny function in a single with executes, in a start
not	def	try	none
or	del	finally	

* assertion → it indicates the warnings

Loops

for printing integers in python

```
x = 20;
for i in range(x):          // o/p: 0, 1, 2, ..... 20
    print i

for i in range(10,x)        // o/p: 10, 11 .... 20.
    print i
```

Ex:- a = ['apple', 'mango', 'banana']
for i in a:
print i

// o/p: apple, mango, banana

Ex:- a = ['apple', 'mango', 'banana']
b = ['red', 'orange', 'blue']
for i in a:
for j in b:
print(i,j)

→ for loop in a for loop.
Two for loops.

Date
Page

while loop:-

Ex: x = 10
i = 0
while (i < x):
    print i
    i++

*raw_input is used to enter input from keyboard

Ex: a = raw_input('I am vinod').
    print a

Ex: input = raw_input("enter your input")
    for x in input:
        print x

o/p: H e l l o W o r l d

if

x = True
y = False
if x and y:
    print "Success"
else:
    print "failure"

Continue:-

x = 20
for i in range (x):
    if i < 3:
        Continue
    else:
        print i

① write a program to findout no.of vowels in a string using python

a = raw_input("Enter the string")
Count = 0
for i in a:
if i == 'a' or i == 'e' or i == 'i' or i == 'o' or i == 'u':
    Count ++

```
print "owels count =" + i
        (or)
a = raw_input("Enter the string")
    count = 0
    owel = "AEIOUaeiou"
    for i in a:
        for i in owel:
            count + +
    print "owels count :" + i
```

ii) How to concatenate two strings in python

```
a - "hello"
b = "world"
c = a+b
print c
```

Functions:

```
def Concat (x):
    b = x + 'ing'
    Print b
a = "helloworld"
Concat(x)
```

```
def add (a, b);
    c = a+b
Print "addition value is
    add(2,3)
```

12/11/18

Ex:- Adding of 2 numbers?

```
def add(a,b):
    c = a+b
    return c
print "The value is " +add(2,3)
```

Ex:- finding factorial of a number?

```
def fact(value):
    if value==0 or value==1:
        return 1
    else
        return fact*fact(value-1)
a = fact(5)
Print "The factorial value is" +a.
```

File Handling in Python

r → is used to read a file if the exists
w → is used to write into file, if file doesn't exists it creates the file.
a → is used to append the data in file, if file doesn't exists it creates the file.

r+ → used to read and write the file. It writes from the starting.
w+ → used to write and read the file. It overwrites the file.
a+ → used to write and read the file. It appends the data in the file. It writes from the last.

open → is used to open a file.
file.close → used to close the file.

Syntax:

```
open(" path", permission)
file.close(file)
```

readlines → used to read the content of the file.

Ex:-
```
def readfile(file):
    fd = open(file, r)
    lines = fd.readlines()
    for line in lines:
        print line
path = raw_input("Enter the path \n")
    readfile(path)
```

Ex:
```
try
        def readfile(file):
    try:
        fd = open(file, r)
    finally:
        print "Cannot open file"
except IOError:
        print "The file does not exist"
path = raw_input("Enter the path \n")
    readfile(path)
```

Ex:- How to print a particular file in a folder.
```
import glob
    os.chdir(path)
    for file in glob.glob(".txt"):
        print file
```
Ex:- for printing any type of file.
```
import os
    for file in os.walk(path)
        print file.
```

Date 13/11/18
Page

6) How to create excel sheet?

import xlrd → To read workbook

import xlwt → To write into workbook

Ex:- xlwt.Workbook() is used to create the excel or spread sheet

wb.add_sheet('Sheet name')

ws.write(Row, Col, data)

wb.save(D://x42/abc.xlsx)

8) How to open existing worksheet?

xlrd.open_workbook(D://x42/abc.xlsx)

Sheet1 = wb.sheet_by_index(0)

Sheet1.cell_value(0,0)

Sheet1.nrows

Sheet1.ncolumns

Ex:

Test filename: x42

Test Case: 1

Test Result: x42

Test Case: 2

Test Result: x42

folder.

if "Test" in line and "Case" in line:

line.split(':', 1)[1]

(1 ↑ number of split; [1] ↘ index)

os.walk is used Search the folders & sub folders

Test result

9) How to extract Data into a spread sheet/excel sheet.

```
import xlwt
import xlrd
import glob
import os
Row=1
wb=xlwt.Workbook() //To create excel sheet
ws=wb.add_sheet("Text Result")
result=raw_input("Enter the Path\n")
    ws.write(0,0,"filename")
    ws.write(0,1,"Text Case")
    ws.write(0,2,"Text Result")
for path, subdirs, files in os.walk(result):
    os.chdir(path) //only for .doc, .txt.
for file in glob.glob("*.txt"): // to extract .doc or .txt
        fd=open(file,"r")
        for line in fd:
if "file" in line and "name" in line and ':' in
        line:
    data=line.split(':',1)[1]
    ws.write(Row,0,data)

elif "Test" in line and "case" in line and ':' in
    line:
        data=line.split(':',1)[1]
        ws.write(Row,1,data)

elif "Test" in line and "Result" in line and ':' in
    line:
        data=line.split(':',1)[1]
        ws.write(Row,2,data)
        Row++.
wb.save("D://xyz/abc.xls") // for saving the workbook
```

Filename: xyz
Text-Case : 1
Text Result: xyz
Text Case : 2
Text Result: Pass

	0	1	2	3
	File	Text Case	Text Result	
1	xyz	1	Pass	
	Sheet1			

Date 21/11/2018
Page

OOPS

1. Encapsulation
2. Polymorphism
3. Inheritence
4. Datahiding

1. Encapsulation:

Combining data and functions in to a single entity is called as encapsulation

2.) Polymorphism:-

Taking more than one form is called as Polymorphism.

3.) Inheritence:-

Inheritence is the property by which objects of one class may aquire the properties of other class.

4. Datahiding:-

Data is used to hide particular data defined inside the class.

Class:- class is a userdefined prototype to create an object. The class Consists of datamembers, methods and class variables.

object:- A object is an instance of a class.

Class variable → A variable that is defined inside a class and can be accessed by the methods and members.

Data member → A data member is a variable defined inside a method and can be accessed by all the class members.

Method:- A method is a special type of function that is defined inside the class.

Function overloading:- Assigning more than one behaviour to a function it called as function overloading

Operator overloading:- Assigning more than one function to an operator it called as operator overloading.

```
Class Employee:
    employee_count = 0
    def __init__(self, name, number):
        self.name = name
        self.number = number
        employee_count ++
    def display(self):
        print "Employee name" + self.name
        print "Employee number" + self.number
    def total_emp_count(self):
        print "Total employee count" + Employee.employee
                                        _count
    A = Employee("Shivi", 1)
    B = Employee("Rajo", 2)
    C = Employee("Vinod", 3)
        A.display()
        B.display()
        C.display()
    C.total_emp_count()
```

del → is used to destroy the object.

```
    def __del__(self):
        print "object destroy"
```

del A
del B
del C

Datahiding

```
Class Employee:
    --count
    def --init--(self):
        Employee--count+=1
    def display(self):
        print Employee.count
A = Employee()
A.display()
Print "Employee--count" //error
```

22/11/18

Inheritance:

Types of Inheritance
1. Single.
2. multiple.
3. multilevel.
4. Hirachical.
5. Hibrid.

1) Single Inheritance:

Accessing the properties from Baseclass to derived class.

Ex:
```
class parent:
    value=0
    def __init__(self):
        Print "invoking parent class\n"
    def setattr(self, data):
        self.data=data
        parent.value = self.data
```

```
    def getattr(self):
        print parent.value
class child(Parent):
    def __init__(self):
        print "Invoking child class".
    c = child()
    c.setattr(200)
    c.getattr().
```

Method overriding:-

```
Ex:- class parent:
    def __init__(self):
        print "In Parent class"
    def fun(self):
        print "Accessing from parent class"
class child(Parent):
    def __init__(self):
        print "In child class"
    def fun(self):
        print "Accessing from child class"
c = child()
c.fun()
```

Method overloading:-

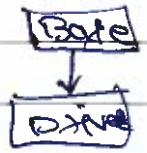
```
class load:
    def __init__(self):
        "The class load is invoked"
    def abcd(self, a, b)
        self.a = a
        self.b = b
```

```
def abcd(self,a)
    self.a=a
l=local()
l.abcd(2)
l.abcd(2,3)
```

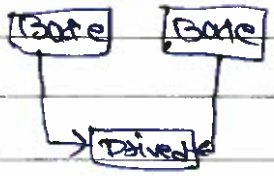
23/4/18 operator overloading:-

```
class operatorplus:
    def __init__(self,a,b):
        self.a=a
        self.b=b
    def __str__(self):
        print self.a, self.b
    def __add__(self,other):
        self.a = self.a + other.a
        self.b = self.b + other.b
a=operatorplus(2,3)
b=operatorplus(3,6)
print a+b
```

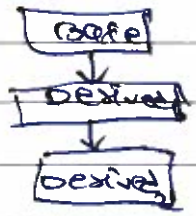
Single:-



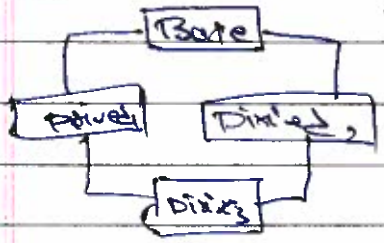
multiple



multiplelevel



Hybrid



55

Date 26/4/18
Page

UDS

UDS (Unified Diagnostic Services) (ISO 14229.)

* UDS is used to find the faults in a vehical. Those faults are stored in EEPROM.

1) Session Control - 10 → Service ID
2) ECU Reset - 11 (Electronic Control Unit).
3) Testes present - 3E
4) Read DID - 22 (DID) → Data Identifier
5) Write DID - 2E
6) Clear DTC - 14 (DTC) (Diagnostic Trouble Code)
7) Read DTC - 19 → Stored in EEPROM
8) Security Access - 27
9) Communication Control - 28
10) Control DTC setting - 85

1) Session Control - 10:-

It has 3 sessions

i. Default Sellion - 01 → subfunction
ii. Programming Sellion - 02
iii. Extended sellion - 03

→ Extended sellion is used to read & write.

→ Default Sellion is used to read the data. But we cannot flash it.

→ Programming Sellion is used to flash

02	10	02	B1	B2	B3	B4
PCI	length	service ID	sub fun	7F	11	01	NRC
				51	01		

02 50 01 → Positive Response

7F 10 01 13 → Negative Response

7F: Indicates the Negative response

13: NRC type

→ Every 6th bit is high for Positive response.

→ 7F will be added to the first bit if it is negative response.

2) ECU Reset:- (Electronic Control Unit)

It has 3 types.

i) Hard Reset - 01

ii) Soft Reset - 03

iii) key on off Reset - 02

Ingation cycle:-

OFF → on → ACCessory - RUN - Running

Specified ECU

3) Tester present:-

The tester present will triggard for every 4 sec. in a particular session.

04 → 50 02 ←

80 3E 04 ↑

3E 80 → alive if we send as tester present Command (or) service ID then the session will not trigger current diagnostic session it will be alive till it complete flash. Then we will get the positive response.

4.) Read DID:- → is for read

22 [1901] → Data identifier

DID NO

62 19 01 XXXXX XXXX XXXX

LDW is a feature to enable LDW we set Bit 4.

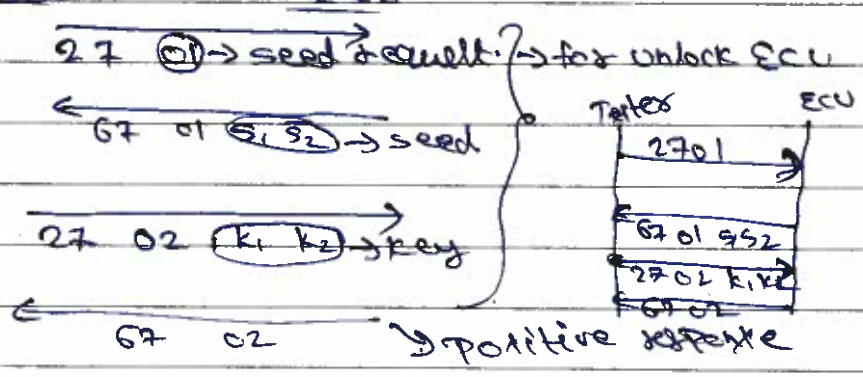
5) write DID:-

2E 19 01 then we should post the data which we seen in read mode

2E 19 01 [0000 0011] [XXXX] [XXXX] [XXXX]

8) Security Access - 27:-

If we want to flash the code, remove code etc, write the data. first we need to take security Access.



If K1 K2 are invalid then it will send 7F 27 02 35

9) Communication Control :- 28

i) Enable TX & RX - 00
ii) Enable Rx Disable Tx - 01
iii) Enable Tx Disable Rx - 02
iv) Disable Both - 03

10) Control DTC Setting - 85:-

i) DTC Setting - on - 01
ii) DTC setting - off - 02

To on DTC setting 85 01
To off DTC setting 85 02

→ Enable & Disable DTC Setting control DTC is used.

7) Read DTC - 19:-

04 → Active → Still the fault is alive
89 → Historic → It came but automatically recovered

19 01 89 01 91 01 1902
DTC no.1 DTC no.2

~~59 89~~

Send 19 89

59 89 all Historic Id's like DTC1, DTC2, DTC3 e

Send 19 01 → all the response

→ ~~All~~ 59 01 → all DTC list like DTC1, DTC2 et

send 19 04 → for all active DTC's

59 04 →

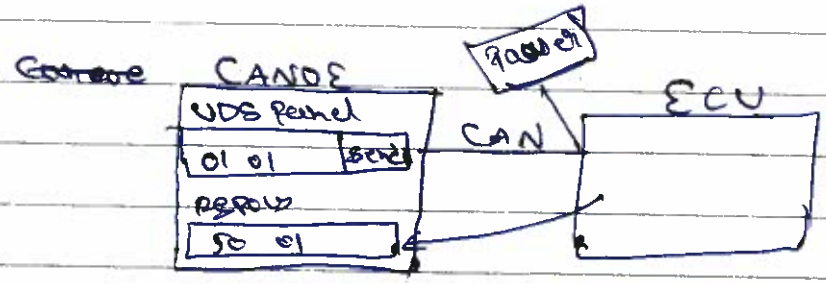
To get whole information about DTC the Command will be 19 02 DTC1 (DTC number)

we Cannot clear Active DTC, we can clear only ~~Historic~~ hiltoric DTC's.

Service ID

~~Command~~ to clear hiltoric DTC - 14 ff ff ff

02/12/2018 → The faults are called as DTC (Diagnostic Trouble code).



*for positive response 40 + SID (Service ID)

*for negative response 7F (1st byte) SID (2nd bytes) Subfun NRC 3rd byte 4th byte

→ Diagnostic Session Control will be alive for 5 sec later if goes to default session.

Date
Page

→ If we want to make it alive for more than 5sec then we need to give tester present sub function ID command

→ TesterPresent is used to trigger the diagnostic session control it makes the diagnostic session control alive till our work gets complete.

Ex:- If data is flashing into ECU and it it takes more than 5sec. it will go into default session it will not stay in programming session. To overcome that we can give testerpresent session ID with time to trigger the diagnostic session control If we enter 4 for every 4sec it will keep triggering.

→ If we give 3E 80 it will not trigger periodically and the session will be alive continuously

Read Data by Identifier:- It is used to modify the data

Read DTC → 19 01

19 [01] 1901 1902
status byte
DTC No1
DTC No2

1901 → all the DTC information will be given.
1903 → for particular DTC information will be given
1904 → will display all the active DTC's
1908 → will display all historic DTC's

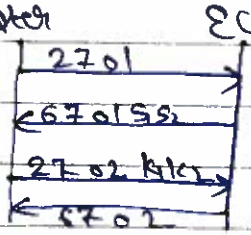
→ Active DTC: The faults which is active and cannot be cleared

→ Historic DTC: The fault has occured but it can be cleared automatically or by Tester.

→ 14 FF FF FF is used to clear all historic DTC's

Security Access

→ Initialy whatever that we need to do with the ECU first we need to take Security access 27 01 : is for Seed request. once we send the seed request we will get response from ECU with seed. with that seed we can calculate the key. and with the calculated key value we can open the ECU and do the access.



Communication Control: is Basicaly used to enable or disable the communication between PC & ECU PC[CANOE] & ECU.

Control DTC setting: If we enable DTC setting then only the DTC's will be logged. If we Disable the DTC setting then the DTC's will not be logged only old faults can be seen.

① How to calculate CAN baudrate & parameters required to calculate it?

② $\frac{\text{clock}}{2 \times (\text{baudrate}+1)} \times 3 + \text{timesegment}_1 + \text{timesegment}_2$

↳ clock sampling pulse

Tseg1 ⇒ before sampling point

Tseg2 ⇒ After sampling point

③ Diff b/w CAN & CANKING?

④ * CAN is the code that is present in the MC.

* CANKING is the simulation s/w to see what is transmitting from one MC to another MC in bits.

⑤ Diff b/w DSP & TMS?

⑥ TMS consists of DSP processors.

⑦ Diff b/w EEPROM & Flash? will CAN support ATMEGA

⑧ * In EEPROM we can replace data in particular memory address

* In flash we cannot replace particular memory address, we should replace the complete data

⑨ ADC?

⑩ formula to calculate Analoge to Digital value.

$\frac{Vin}{Vout} \times V_{ref\ voltage}$

(Max value)

for code:

→ Start Conversation

→ wait till Conversion completes

→ Read the 12bit or 10bit ADC from register