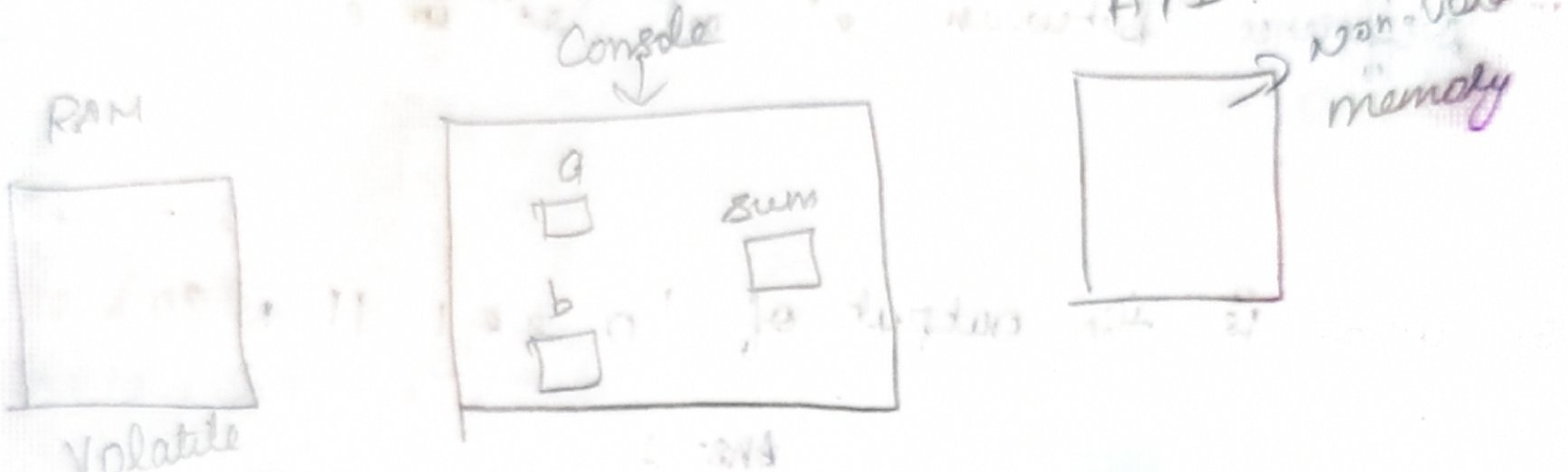
21.07.25

File Handling :-



When the PC is off / on the all files are gone.

It contains Three mode

r → read the file data

w → write a file

a → write

Difference between w & a w → per create a new file and write. a → it point the old file data to add something here.

write mode → To delete the pre-existing data and write a new data.

append mode → It pre-existing data in the same file and to add the new data.

Read mode (r) :-

fgetc(); getc() e → To point character to get the input from the user.

fgets(); gets() ⇒ S → The S to get the input from the user using string.

①

test.txt

Hello World

```
#include main()
{
FILE *fp;
char c;
fp=fopen("test.txt","r");
c=fgetc(fp);
Printf("%c",c);
c=fgetc(fp);
Printf("%c",c);
}
```

O/P :-

He

How the next print e because the file pointer automatically to move the next character.

```
int main()
{
FILE *fp;
char c[100];
fp = fopen("test.txt","r");
fgets(c, 5, fp);
Printf("%s",c);
fclose(fp);
}
```

Write Mode (w):

fputc(); // to write a single character at a time

fputs(); // to write a many characters

In case to read & write in binary files we use also the extension bin

"wb", "wb", "ab"

(2).

```
int main()
{
  FILE *fp;
  fp = fopen("test.txt", "w");
  fputc('c', fp);
  fputs("Hello welcome", fp);
  fclose(fp);
}
```

already build in the header file stdio.h

FILE → It is a file data type.

3.

21.07.25

ftell():-

↳ It is used to find out the file pointer in the file with respect to starting of the file.

Syntax:-

long in ftell(FILE *stream).

fseek():-

fseek() used to set the file of pointer.

like file ∞.

Hello this is Aparnika

↳

=> It is used to modify the file content.

Syntax :-

int fseek (~~File~~ FILE * stream , ~~long~~ long int ~~offset~~ ,
int whence).

⇒ ~~BA~~ SEEK_SET
SEEK_CUR
SEEK_END.

(3)

```
int main()
{
FILE *fp;
fp = fopen ("files5.txt", "a");
if ( fP== NULL)
    Printf ("File is not found");

else
{
printf ("ftell = %ld \n", ftell(fp)); //
fseek ( fP, 2, SEEK_SET);
```

→ no point to file location

→ Second position la file pointer set panni

```
Printf ("ftell = %ld \n", ftell (fP));
```

↳ location patha.

Set panna place la Poi-thu file location count Panni solluvan

```
char ch;
 ch = fgetc (fP);
  Printf ("ch = %c \n", ch);
  fclose (fP);
}
}.
```

SEEK_CUR
↳ It will find current position to Destination.

Rewind ().

Syntax :-

void rewind (FILE * stream).

Rewind ()
↳ It is used to point the beginning position

Eg :- I am in a ~~the~~ position file pointer I move to first position to use rewind (fp); to move the beginning.

Modes of File :-

r w a r+ w+ a+ rb wb ab
                    rb+ wb+ ab+

read binary.

fprintf () :-

Syntax:-

fprintf (FILE *stream, "const char* format", list of variables);

Ex:-

fprintf (fp, "%d %d %s", id, num, str);

fscanf (fp, "%d %s %d", &id, &name, &salary);

fwrite (), fread () :-

fwrite()

part of the block of memory to be written

number of element to be written.

Syntax:-

size_t fwrite (void * ptr, size_t size, size_t nmemb, FILE * stream)

Size of each element to written.

```
int main()
{
  FILE *ptr;
  fp = fopen ("test.bin", "wb");
  int arr[4] = {1, 2, 3, 4};
  fwrite (&arr, 8, 2, fp);
  return 0;
}
```

```
fread (&arr, sizeof (arr), 1, fp);
Printf ("%d %d %d %d", arr[0], arr[1],
                       arr[2], arr[3]);
```

File operations in C :-

4. Check if file exists before opening.

```
int main()
{
    FILE *fp;
    const char *filename = "data.txt";
    fp = fopen(filename, "r");
    if (fp == NULL){
        printf("File %s does not exist or cannot
                opened", filename);
    }
else
{   printf("File %s exists and opened successfully",
                filename);
    fclose(fp);
}
```

5. count number of lines in a given files.

```
int main()
{
  FILE *fp;
  char ch;
  int lines = 0;
  fp = fopen("file.txt", "r");
while ((ch = fgetc(fp)) != EOF){
    if (ch == '\n')
    {
      lines++;
    }
}
```

```
    fclose (fp);
    printf (" total number of lines in file is
                                %d \n", lines);

    return 0;
}
```

6. Copy contents of one file to another.

```
int main()
{
  FILE *src, *dest;
  char ch;
  src = fopen ("file.txt", "r");
  dest = fopen ("Main.txt", "w");
while ((ch = fgetc(src)) != EOF)
  { fputc (ch, dest);
    }
printf ("File copied successfully ");
 fclose (src);
 fclose (dest);
 return 0;
}
```

7. DIY

7. Difference between text and binary file Handling.

Features	Text File	Binary File :
Data Format	Human - readable characters	Raw bytes (0,1) (machine readable)
Extension	.txt , .csv, .log	.bin, .dat , .exe
Storage	Data is stored as string	Data is stored in exact binary form.
File mode	"r", "w", "a"	"rb", "wb", "ab".
Size	Generally larger (because of formatting).	More compact.

8. How to append data to an existing file.

```
int main()
{
  FILE *fp;
  fp= fopen("main.txt", "a");
  fputs("PHYTEC", fp);
  fclose(fp);
}.
```

9. Use fscanf() to read formatted input from file.

```
int main()
{
  FILE *fp;
  char name[50];
  int age;
```

```
fscanf (fp, "%s %d", name, &age);
printf ("%s in %d", name, age);
fclose (fp);
return 0;
}
```

10. Write a program to reverse the content of a file.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
FILE *infile, *outfile;
char ch;
long filesize;
char *buffer;
int i;
infile = fopen ("main.txt", "r");
fseek (infile, 0, SEEK_END);
filesize = ftell (infile);
rewind (infile);
buffer = (char *) malloc (filesize + 1);
fread (buffer, sizeof (char), filesize, infile);
fclose (infile);
outfile = fopen ("File.txt", "w");
```

```
for (i = filesize-1 ; i>=0 ; i--)
  {
    fputc ( buffer[i], outfile);
  }
printf ("File content reversed successfully");
fclose (outfile);
free (buffer);
return 0;
}
```

28.07.25

Message Queue :-

Message Queue is a Inter process communication.

That allows processes to exchange data by sending and receiving message in structured, asynchronous way.

msgsnd → will initialize the queue.

msgrcv → will be used to receive the message.

msgctl → control action that is inclusive of deleting the Queue.

fzk → file to key.

key_t → message send.

key = ftok (. "file name", 'A', or 'a', 'b') ;-

perror ("key not generate");

message id to need.

mqd_t → It is a Message Queue Discriptor.

0660 → It is file permission like read/write for all onfile file.

11. Create and open a message queue using 'mq_open()'

mqd_t mq = mq_open ("/myqueue", O_CREAT | O_RDONLY, 0644, &attr);

12. Send a message using 'mq_send()' in c?

mq_send() function sends to message to Message Queue. It takes a message discriptor, message buffer, message length, Priority argument.

Syntax:-

int mq_send (mqd_t mqdes, const char *msg_ptr, size_t msg_len, unsigned int msg_prio);

Ex:- mq_send (mq, "Hello Physec", 6, 0).

12) Receive Message using mq_receive();

mq_receive() - this function used to retrieve the message from a specified queue.

Syntax :-
ssize_t mq_receive(mqd_t mqdes,
char *msg_ptr, size_t msg_len,
unsigned int *msg_prio);

Example :- mq_receive(mq, buffer, MAX_SIZE,
NULL);

14. Set attributes of a message queue?

message queue attributes are.

mq_flag	it check blocking / non-blocking mode.
mq_maxmsg.	maximum number of message the queue can hold.
mq_msgsize	Maximum size of each message.
mq_curmsg	current number of messages in the queue.

Why message Queue :-

↳ The message queue need is when two or more processes need to exchange data without sharing memory.

message queue in communication between.

↳ process can be completely independent

↳ one process sends a message.

Another receive it.

→ sender & receiver do not need to be active at the same time.

→ Message are stored in the queue until the receiver picks them up.

→ Order is Maintained.

15) Write a program to demonstrate blocking receive.

```
#define Queue_Name "/test_queue"
#define MAX_SIZE 1024.
int main()
{
 mqd_t mq;
 char msg[MAX_SIZE];
mq=mq_open(Queue_name, O_WRONLY);
if( mq == (mqd_t)-1){
  perror ("mq_open");
   exit (EXIT_FAILURE);
}
. printf("Enter message to send");
 fgets (msg, MAX_SIZE, stdin);
 msg [strcspn (msg, "\n")]='\0';
if (mq_send (mq, msg, strlen(msg),0) == -1) {
perror & ("mq_send");
 exit (EXIT_FAILURE);
  }
```

This is sender side

```
Printf ("Message sent:%s
        \n", msg);
mq_close(mq);
 return 0;
}.
```

This is a receiver side

29.07.25

IPC (Inter process communication)
(Pipes, Shared Memory, Semaphores).

8). Write a program to communicate using unnamed Pipes.

IPC stands for Inter-process communication.
It refers to the mechanism that allow Process to communicate with each other.

Process are independent and have a separate memory space, so they can't share data directly. IPC provide tools for them to exchange data or signals.

→ Pipes
→ Message Queue
→ Shared Memory
→ semapholes
→ Sockets.

Pipe :-
The Pipes allows two or more processes to communicate with each other by creating a unidirectional or bidirectional channel between them.

A Pipe is a virtual communication channel that allows data transfer between the processes., either one-way or two-way. Pipes can be implemented using system calls.

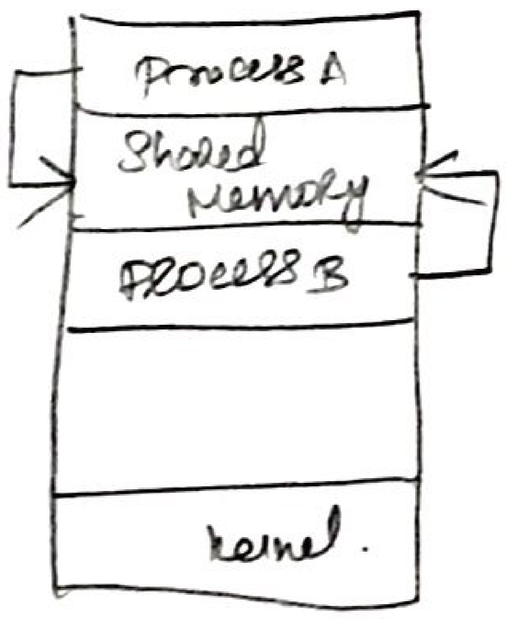
Pipes :
↳ file discriptor is key. the key is access to a file. (where to read & write.

Co-operating process to depends the another process.

Co-operating process to implement in Ipc

→ Shared Memory
→ Message passing

Shared memory :-



TWO PROCESS two share the one memory for read & write.

Shared memory to explain in the Producer and Consumer problem to explain the others.

↳ To process to need a one buffer both (produce, consume)

↳ Share this.

unbounded buffer → no limit on the size

bounded buffer → fixed size.

The consumer may have to wait for new items, but the producer can always produce the new item.

⇒ Normal, the OS does not allow one Process to access the memory of another process.

⇒

For ex:- A compiler may produce the assembly code, which is consumed by an assembler.

⇒ To allow producer and consumer processes to run concurrently,

buffer → It is shared by producer, consumer.

The producer to produce one item & keep it the buffer and the consumer consuming another item from the same buffer.

Shared Memory Functions:-

ftok() → It generates the unique key for shared memory.

shmget() → Allocates or retrieves a shared memory segment.

shmat() → Attaches the shared memory segment to the process's address space.

shmdt() → Detaches the shared memory segment from the process.

shmctl() → Performs control operations on the shared memory segment.

Q4) Create and attach shared memory segment using 'shmget()', and 'shmat()'.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <string.h>
int main()
{
    key_t key = ftok("shmfile", 65);   // → This key to generate the id for shared memory.
    int shmid = shmget(key, 1024, 0666, IPC_CREAT);   → to get the id from the key
    if (shmid == -1) {   // This condition to check shared memory created or not. to
        perror("shmget");                                  check it.
        exit;
    }
    char *str = (char *)shmat(shmid, NULL, 0);   → attach to source the shmid.
    if (str == (char *)(-1)) {
        perror("shmat");
        exit(1);
    }
    printf("Writin. to shared memory:---");
    strcpy(str, "Hello from shared memory");
    printf("Reading from shared memory: %s", str);
    shmdt(str);
    return 0;
}
```

35). write and read data using shared memory.

TO Same code for 34 Question.

36). Detach and Delete shared memory using 'shmdt()' and 'shmctl()'.

```
int main()
{
key_t key = ftok("shmfile", 65);
int shmid = shmget(key, 1024, 0666 | IPC_CREAT);
if(shmid < 0){
    perror("shmget failed");
    exit(1);
}
char *str = (char *) shmat(shmid, NULL, 0);
if(str == (char *)(-1)){
perror("shmat failed");
exit(1);
}
strcpy(str, "Hello from Detach the Shared memory data");
printf("Data read from shared Memory: %s\n", str);
if(shmdt(str) == -1){
        perror("Detach failed");
        exit(1);
} else{
    printf("Shared Memory detached successfully");
} }
```

```
if (shmctl (shmid, IPC_RMID, NULL) == -1)
    { perror ("shmctl failed");
      exit (1);
    }
else { printf ("Shared Memory Deleted successfully.");
    }
return 0;
}
```

IPC_RMID → Inter process communication - Remove Identified.

↳ remove the shared memory segment from the system.

Semaphores :-

Semaphores is used to the process synchronization and controlling access to shared resource. They are implemented as a system resource within the kernel.

One line on semaphore - Allow only one process to access shared memory data at a time (mutual exclusion)

Mutexs → like semaphores, but typically used for thread.

Types of semaphore :-

↳ Binary semaphore

↳ Counting semaphore

Binary Semaphore :-

Functin like Mutexs, having two states (0 or 1) representing lock or unlock state.

Counting Semaphore :-

This maintain non-negative integer count. This count represents how many instances of a resource are available.

operations

semop() → the semaphore set, include waiting for a semaphore to become available (decrementing its value). and signaling that a resource is free (incrementing its value).

semget() → used to create or get an existing semaphore set.

semctl() → Used for control operations on a semaphore set, such as setting initial values or removing the set.

3). Synchronize access to shared memory using semaphores.

```
int main()
{
```

```
union semun {
    int val;
};
void sem_wait (int semid)
{ struct sembuf sops = {0, -1, 0};
    semop (semid, &sops, 1);
}
void sem_signal (int semid){
    struct sembuf sops = {0, 1, 0};
    semop (semid, &sops, 1);
}
int main() {
key_t shm_key = ftok ("shmfile", 65);
key_t sem_key = ftok ("semfile", 75);
int shmid = shmget (sem_key, 1024, 0666 | IPC_CREA
char *data = (char *) shmat (shmid, NULL, 0);
int semid = semget (sem_key, 1, 0666 | IPC_CREAT);
union semun u;
 u.val = 1;
semctl (semid, 0, SETVAL, u);
pid_t pid = fork();
if (pid == 0) {
    sem_wait (semid);
    printf ("child writing a Shared Memory");
    strcpy (data, "child say hello");
    sleep(2);
```

```
printf("child done");
sem_signal(semid);
}
else {
    sem_wait(semid);
    printf("parent waiting to shared Memory");
    strcpy(data, "parent say hi");
    sleep(2);
    printf("Parent done");
    sem_signal(semid);
}
shmdt(data);
if(pid >0) {
    sleep(3);
    shmctl(shmid, IPC_RMID, NULL);
    semctl(semid, 0, IPC_RMID);
}
return 0;
}
```

39) Compare pipes vs Message ... Memory.

Feature	Pipe	Message Queue	shared Memory.
Type	Byte-stream based	Message based	Memory based
Direction	Unidirectional (Pipe) bidi (FIFO)	Bidirectional	Bidirectional
Speed	Slow	Moderate	Fastest.
kernel Involvement	Yes (buffered in kernel)	Yes	direct user Space access.
Max data size	limited by pipe buffer	Max-message Size	larger.
Data Structure	Linear byte Stream	Messages with ID.	Memory block.

40) What are the use case of each IPC mechanism?

Pipes :-

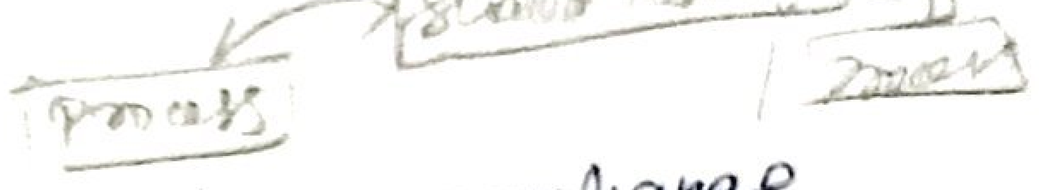
Use case :- When two related processes need to exchange simple data.

Example : A Parent process can send commands to a child process using pipe().

Messag Queue :

Use case :- when multiple processes need to exchange structured messages asynchronously

Ex :- A logging service collecting logs from different application using msgsend() / msgrcv().



shared memory :-

Use-case :- when processes need to exchange large amount of data quickly, like real-time audio or sensor data.

Ex :- A producer writes camera frames to shared memory, and a consumer processes them for object detection.

Semaphores :-

use-case :- used a alongside shared memory on files to ensure mutual exclusion.

Ex :- Multiple processes updating a shared counter with semop() to lock / unlock access.

17. Delete a message queue with "mq-unlink()"?

mq-unlink(). — this functions remove a named posix message queue from the system.

Syntax:-

```
#include <mqueue.h>  // this header contains the posix message contain all built in functions.
int mq_unlink(const char *name);
                      └→ queue name.
```

```
#define Queue_name "/my-Queue".
#define MAX_SIZE 1024.
int main() {
  mqd_t mq;
  struct mq_attr attr;
  char buffer[MAX_SIZE];
  attr.mq_flag = 0;
  attr.mq_maxmsg = 10;
  attr.mq_curmsgs
mq = mq_open(Queue_Name, O_CREAT | O_RDONLY, 0644, &attr);
if(mq == (mqd_t)-1) {
    perror("mq_open");
    exit;
}
printf("waiting for message");
ssize_t byte_read = mq_receive(mq, buffer, MAX_SIZE, NULL);
if(bytes_read < 0) {
    perror("mq-receive");
```

19. Monitor a message queue using 'select()' or 'poll()'?

select() → allows you to monitor multiple file descriptor (Eg. sockets, pipes, message queue) at the same time.

↳ Data is ready to read

↳ space is available to read.

↳ An exception occurs.

POSIX Message Queue don't behave like normal file discriptor, so can't directly use select() or poll() on them in standard way.

If need to monitor the message-queue, use like a mq_notify() to ask the kernel to notify us when a message arrives. then to request file discriptor notification using SIGEV_Thread.

— Pin complete.

20. Create a producer and consumer using POSIX MQ.

Producer.c.

```
#define Queue_name "/my_Queue"
#define MAX_SIZE 1024.
int main()
{
```

```
char msg[MAX_SIZE];
mq = mq_open(queue_name, O_WRONLY | O_CREAT,
                                0644, NULL);
if(mq == (mqd_t) -1){
    perror("Queue creation & open failed");
    exit(EXIT_FAILURE);
}
while(1)
{
 printf("Enter message");
 fgets(msg, MAX_SIZE, stdin);
 msg[strcspn(message, "\n")] = '\0';
 if(strcmp(msg, "exit")==0) {
    break;
 }
if(mq_send(mq, msg, strlen(msg), 0) == -1) {
    perror("Message send");
 }else
    {
     printf("Message sent: %s\n", msg);
    } }
mq_close(mq);
return 0;
}
```

consumer.c

```
#define Queue_name "/myqueue"
#define MAX_SIZE 1024
int main() {
mqd_t mq;
mq = mq_open( char buffer[MAX_SIZE + 1];
ssize_t bytes_read;
struct my_attr. attr = {
    . mq_flag = 0;
    . mq_maxmsg = 10,
    . mq_msgsize = MAX_SIZE;
    . mq_curmsgs = 0.
};
mq = mq_open(Queue_name, O_CREAT | O_RDONLY,
                                    0644, &attr);
if (mq == (mqd_t)-1) {
    perror("mq_open");
    exit;
}
Printf("consumer is waiting for message");
while(1) {
    bytes_read = mq_receive(mq, buffer, MAX_SIZE,
                                    NULL);
    if (bytes_read >= 0) {
        buffer[bytes_read] = '\0';
        printf("received : %s \n", buffer);
    if (strcmp(buffer, "exit") == 0) {
        printf("Exit command received close");
        break;
    }
}
```

```
    else
      {
        perror ("mq_receive");
      }
    }
  mq_close (mq);
  mq_unlink (queue_name);
   return 0;
}
```

System V message in the traditional method of IPC Model

21. create message queue with 'msgget()'?

```
int main()
{
   key_t key;
   int msgid;
   key = ftok (".progfile", 65);
   msgid = msgget (key, 0666 | IPC_CREAT);
   if (msgid == -1)
        {
          perror (" Error");
          exit;
        }
   printf (" Message Queue ID : %d \n", msgid);
   return 0;
}
```

msgget (key_t char *pathname, int proj_id);

→ arbitary id.

22) send and receive using msgsnd() and msgrcv():

Sender.

```
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <string.h>
#define MAX_TEXT 100
struct msg_buffer {
    long mtype;
    char mtext[MAX_TEXT];
};
int main()
{
    key_t key = ftok("profile", 65);
    int msgid = msgget(key, 0666 | IPC_CREAT);
    Struct msg_buffer msg;
    msg.mtype = 1;
    Strcpy(msg.mtext, "Hello from sender");
    msgsnd(msgid, &msg, sizeof(msg.mtext), 0);
    printf("Sent: %s \n", msg.mtext);
    return 0;
}
```

Receiver :-

```
#define MAX_TEXT 100
Struct msg_buffer {
    long mtype;
    char mtext[MAX_TEXT];
};
int main()
{
```

```
key_t key = ftok("progfile", 65);
int msgid = msgget(key, 0666 | IPC_CREAT);

struct msg_buffer msg;

msgrcv(msgid, &msg, sizeof(msg.mtext), 1, 0);
printf("Received : %s \n", msg.mtext);
msgctl(msgid, IPC_RMID, NULL);

return 0;
}
```

01.08.25.

23) Write a program to exchange message between two process.

```
#define MAX_SIZE 100

struct msg_buffer {
    long mtype;
    char mtext[MAX_SIZE];
};

int main()
{
    key_t key;
    int msgid;

    key = ftok("msgfile", 65);

    msgid = msgget(key, 0666 | IPC_CREAT);

    if (msgid == -1) {
        perror("msgget");
        exit(1);
    }
```

```
pid_t pid = fork();
if (pid<0) {
    perror("fork");
    exit(1);
  }

if(pid>0) {
  struct msg_buffer msg;
  msg.mtype = 1;
  strcpy(msg.mtext, "Hello child, this is parent");
  msgsend(msgid, &msg, sizeof(msg.mtext), 0);
  printf("Parent send : %s \n", msg.mtext);
  msgrcv(msgid, &msg, sizeof(msg.text), 2, 0);
  printf("parent received : %s \n", msg.mtext);
    msgctl(msgid, IPC_RMID, NULL);
  }
else
  {
   struct msg_buffer msg;
   msgrcv(msgid, &msg, sizeof(msg.mtext), 1, 0);
   Printf("child receive : %s \n", msg.mtext);
   msg.mtype = 2;
   strcpy(msg.mtext, "Hi parent, message received");
   msgsnd(msgid, &msg, sizeof(msg.mtext), 0);
   Printf("child send : %s \n", msg.mtext);
  }
  return 0;
}
```

24) Remove message queue using 'msgctl'?

msgctl() - is used to control the message queue.

Prototype :-

```
int msgctl (int msgid, int cmd, struct msqid_ds *buf);
```

```
#include<stdio.h>
#include<sys/ipc.h>
#include<sys/msg.h>
int main()
{
    key_t key = ftok("msgfile", 65);
    int msgid = msgget(key, 0666 | IPC_CREAT);
    if(msgid == -1) {
        perror("message get failed");
        exit(1);
    }
    if(msgctl(msgid, IPC_RMID, NULL) == -1)
    {
        perro("message control failed");
        exit(1);
    }
    Printf("message queue removed successfully");
    return 0;
}
```

25) check for message queue limits?

Message queue limits to find have a two approaches.

View all Ipc limits :-

→ Ipcs -l

check /proc/sys/kernel

cat /proc/sys/kernel/msgmax

cat " /msgmnb

cat /proc/sys/kernel/msgmin.

If you want to change the message queue limits temporarily.

→ sudo sysctl -w kernel.msgmax = 16384

→ sudo sysctl -w kernel.msgmnb = 65536.

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

int main(){
    key_t key = ftok("msgfile", 65);
    int msgid = msgget(key, 0666 | IPC_CREAT);
    struct msqid_ds info;
    if(msgctl(msgid, IPC_STAT, &info) == -1){
        perror("message control failed");
        return 1;
    }
    printf("Max bytes in queue : %lu \n", info.msg_qbytes);
    printf("current Bytes in queue : %lu \n", info.msg_cbytes);
    printf("Number of message in queue : %lu \n", info.msg_qnum);
    return 0;
}
```

26). How to send structured message with System V MQ?

Define message structure.

The core of sending structured messages is to define a C struct that will hold your data; The first number of this struct must be a long type, which represents the message type.

```
Struct my-message{
    long mtype;
    int data1;
    char text[1024];
};
```

Get a Message Queue

Before sending, need a message Queue. Create a new one or get an existing one using msgget().

```
key-t key = ftok("my-key-file", 'A');
int msgid = msgget(key, IPC-CREAT | 0666).
```

Populate the Message Structure

Creat an instance of your message structure and populate its fields, including the mtype and your custom data.

```
Struct my-message msg;
msg.mtype = 1;
msg.data1 = 123;
strcpy(msg.text, "Hello, Structured Message");
```

Send the Message :-

msgsnd() → function to send your structured message to the queue.

```
int msg_size = sizeof(struct my_message) - sizeof(long);
msgsnd(msqid, &msg, msgsize, 0);
```

27). Handle IPC_NOWAIT flag in receive :-

the IPC_NOWAIT flag, when used with the msgrcv() function in System V message Ques.

IPC_NOWAIT is not specified or set to 0, the msgrcv() function will block until a message of specific type is placed on the queue or the message queue is removed.

IPC_NOWAIT(). is set non-zero, msgrcv() will not block if a message of the specified type is available, it will be received. If no msg available the function will return immediately with an error specifically, errno will be set to ENOMSG.

28) Differentiate system V vs POSIX Queue :

Feature	System V message Queue	POSIX message Queues.
API functions	msgget(), msgsnd(), msgrcv(), msgctl().	mq_open, mq_send(), mq_receive(), mq_close.
Identifier type.	Integer Queue ID (int)	Queue name string (const char *).

Naming	Anonymous / key-based (ftok())	Name like files (eg. "/myqueue")
File system Integration	Not visible in file system	Appears under /dev/mqueue
Standard	Old system V IPC	POSIX.1b (real-time extension)
Priority support	No message priority	Yes high priority message delivered first.
Notification mechanism	Not Support	Support signal or Thread_notification through mq_notify().
Access control	Permission through msgctl()	File like permission (mq_open with mqt_t).
Maximum Size	Both Fixed, System-wide (/proc/sys/kernel/msgmax).	Configurable per Queue.
Queue Limits	System-wide (msgmni, msgmax)	//
Thread Safety	Not Inherently Thread safe	Designed for use with threads-

29). Blocking Behaviour in System V MQ.

System V message queues, function like msgsnd() and msgrcv() are blocking by default. This means the process will wait until it can send or receive a message. If I want the operation to return immediately instead of waiting, I use the IPC_NOWAIT flag. This make the non-blocking so, it will return error if it can't proceed.

30. How to Handle the permission error in MQ.

When a process tries to access a message Queue without the necessary rights the Permission will occur.

The usually the permission error happens to call like msgget() → to Create / access.

msgsnd()

msgrcv()

msgctl()

errno = EACCES. // permission denied

You don't have a access to this message Queue.

each message Queue have to kernel maintains a data structure msq ID and first element.

IPC_Perm.