📘 **C Programming Basics**

Lab Session: System Call Timing and Process Synchronization

Course: Operating Systems
Date: 30.05.2025

Program 1: Inter-Process Communication using Message Queue (Sender)

```
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#define MAX 10

struct mesg_buffer {
    long mesg_Type;
    char mesg_text[100];
} message;

int main() {
    key_t key;
    int msgid;

    key = ftok("send", 1109);
    msgid = msgget(key, 0666 | IPC_CREAT);
    message.mesg_Type = 1;

    printf("Write data: ");
```

```
    fgets(message.mesg_text, MAX, stdin);

    msgsnd(msgid, &message, sizeof(message), 0);
    printf("Data sent is: %s\n", message.mesg_text);

    return 0;
}
```

Fedora Output:

```
[2021ict108@fedora ~]$ vi sender.c
[2021ict108@fedora ~]$ gcc sender.c -o sender
[2021ict108@fedora ~]$ ./sender
Write data: Hello
Data sent is: Hello
```

Explanation:

- This program sends data using System V message queues.
- ftok() creates a unique key.
- msgget() creates or accesses a message queue.
- msgsnd() sends the message from sender to the queue.

Program 2: Inter-Process Communication using Message Queue (Receiver)

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

struct mesg_buffer {
    long mesg_Type;
```

```
    char mesg_text[100];
} message;

int main() {
    key_t key;
    int msgid;

    key = ftok("send", 1109);
    msgid = msgget(key, 0666 | IPC_CREAT);

    msgrcv(msgid, &message, sizeof(message), 1, 0);

    printf("Data received is: %s\n", message.mesg_text);

    msgctl(msgid, IPC_RMID, NULL);

    return 0;
}
```

Fedora Output:

```
[2021ict108@fedora ~]$ vi receiver.c
[2021ict108@fedora ~]$ gcc receiver.c -o receiver
[2021ict108@fedora ~]$ ./receiver
Data received is: Hello
```

Explanation:

- This program receives data sent via message queue.
- msgrcv() retrieves the message from the queue.

- msgctl() is used to remove the message queue after reading.
- Works in tandem with the sender program.