Linux IPC Practice Programs: Pipes, Message Queues, and Shared Memory

This 2-hour lab introduces key Inter-Process Communication (IPC) mechanisms in Linux: Unnamed Pipes, Message Queues, and Shared Memory. Each section provides examples and tasks for hands-on learning.

Part 1 – Unnamed Pipe (Approx. 35 min)

Objective: Understand how two processes communicate using a pipe.

Key Topics:
- Creating a pipe using pipe()
- Writing and reading data
- Communication between parent and child using fork()

Example Program (pipe_example.c):

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>

int main() {
    int fd[2];
    pid_t pid;
    char msg[] = "Hello from parent!";
    char buffer[100];

    pipe(fd); // Create pipe

    pid = fork();
    if (pid == 0) { // Child process
        close(fd[1]); // Close write end
        read(fd[0], buffer, sizeof(buffer));
        printf("Child received: %s\n", buffer);
        close(fd[0]);
    } else { // Parent process
        close(fd[0]); // Close read end
        write(fd[1], msg, strlen(msg) + 1);
        close(fd[1]);
    }
```

```
    return 0;
}
```

Task: Modify the program so that the child sends a reply back to the parent.

Part 2 – Message Queue (Approx. 40 min)
Objective: Exchange messages between independent processes using System V message queues.

Key Topics:
- msgget(), msgsnd(), msgrcv(), msgctl()
- Key generation using ftok()

Sender Program (msg_sender.c):

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

struct msg_buffer {
    long msg_type;
    char msg_text[100];
} message;

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

    message.msg_type = 1;
    printf("Enter message: ");
    fgets(message.msg_text, sizeof(message.msg_text), stdin);

    msgsnd(msgid, &message, sizeof(message), 0);
    printf("Message sent: %s\n", message.msg_text);

    return 0;
}
```

Receiver Program (msg_receiver.c):

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

struct msg_buffer {
    long msg_type;
    char msg_text[100];
} message;

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

    msgrcv(msgid, &message, sizeof(message), 1, 0);
    printf("Message received: %s\n", message.msg_text);

    msgctl(msgid, IPC_RMID, NULL); // delete queue
    return 0;
}
```

Task: Enhance this program for two-way communication (chat between two processes).

Part 3 – Shared Memory (Approx. 45 min)
Objective: Understand how processes share data using shared memory segments.

Key Topics:
- shmget(), shmat(), shmdt(), shmctl()
- Synchronization basics using sleep()

Writer Program (shm_writer.c):

```
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <string.h>

int main() {
    key_t key = ftok("shmfile", 65);
```

```
    int shmid = shmget(key, 1024, 0666 | IPC_CREAT);

    char *str = (char*) shmat(shmid, NULL, 0);
    printf("Write Data: ");
    fgets(str, 100, stdin);

    printf("Data written in memory: %s\n", str);
    shmdt(str);
    return 0;
}
```

Reader Program (shm_reader.c):

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

int main() {
    key_t key = ftok("shmfile", 65);
    int shmid = shmget(key, 1024, 0666 | IPC_CREAT);

    char *str = (char*) shmat(shmid, NULL, 0);
    printf("Data read from memory: %s\n", str);

    shmdt(str);
    shmctl(shmid, IPC_RMID, NULL);
    return 0;
}
```

Task: Modify to support bidirectional communication using two shared memory segments.

Summary

IPC Type	Functions Used	Communication Type
Unnamed Pipe	pipe(), read(), write(), fork()	Parent–Child
Message Queue	msgget(), msgsnd(), msgrcv(), msgctl()	Process–Process

Shared Memory	shmget(), shmat(), shmdt(), shmctl()	Fastest, Shared