MINOR ASSIGNMENT-11

Inter-Process Communication: Shared Memory & Message Queue

1. Write a program to create a shared memory segment of size 10 bytes. Make 4 attachments to the shared memory segment to the address space of the calling process and print the number of attachments using the structure filed *number of current attachments* present in the structure **shmidds** defined in the header **<sys/shm.h>**. Check the number of attachment using the shell provided command **ipcs -m**.

```c
#include <stdio.h>
#include <sys/shm.h>
#include <sys/stat.h>

int main() {
    int shmid = shmget(IPC_PRIVATE, 10, IPC_CREAT | 0666);
    if (shmid == -1) {
        perror("shmget failed");
        return 1;
    }

    for (int i = 0; i < 4; i++) {
        void *addr = shmat(shmid, NULL, 0);
        if (addr == (void *)-1) {
            perror("shmat failed");
            return 1;
        }
    }

    struct shmid_ds shminfo;
    shmctl(shmid, IPC_STAT, &shminfo);
    printf("Number of attachments: %lu\n", shminfo.shm_nattch);

    return 0;
}
```

Explanation: Creates shared memory, attaches it four times, and prints the attachment count.

2. Create a C code named shmwriter.c to create a shared memory segment of integer size and store 500 to the segment. Create another program named shmreader.c to access the stored value from the shared memory segment and display it. Let the shmreader.c update the value to 600. Now update the shmwriter.c code to get the updated value and display it. You are not allowed to use semaphore.

Writer Code (`shmwriter.c`):

```c
#include <stdio.h>
#include <sys/shm.h>
#include <sys/stat.h>

int main() {
    int shmid = shmget(1234, sizeof(int), IPC_CREAT | 0666);
    int *data = (int *)shmat(shmid, NULL, 0);
    *data = 500;
    printf("Writer: Stored 500\n");
    return 0;
}
```

Reader Code (`shmreader.c`):

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

int main() {
    int shmid = shmget(1234, sizeof(int), 0666);
    int *data = (int *)shmat(shmid, NULL, 0);
    printf("Reader: Read value %d\n", *data);
    *data = 600;
    printf("Reader: Updated value to 600\n");
    return 0;
}
```

3. Create 2 processes using **fork()**. The child will sent a number to parent using shared memory segment. The parent willl display the received number and doubles it and sends back to the client.
   The client will display the received number.

```c
#include <stdio.h>
#include <sys/shm.h>
#include <sys/wait.h>
#include <unistd.h>

int main() {
    int shmid = shmget(IPC_PRIVATE, sizeof(int), IPC_CREAT | 0666);
    int *data = (int *)shmat(shmid, NULL, 0);

    if (fork() == 0) {
        *data = 10;
        printf("Child: Sent %d\n", *data);
    } else {
        wait(NULL);
        printf("Parent: Received %d\n", *data);
        *data *= 2;
        printf("Parent: Doubled value to %d\n", *data);
    }
    return 0;
}
```

Explanation: Child sends a value, and the parent doubles it.

4. Design a C code to create a message queue and add 4 messages to the queue. Create a receiver code to receive all the messages from the queue till the queue is empty.

Sender Code:

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

struct msgbuf {
    long mtype;
    char mtext[100];
};

int main() {
    int msgid = msgget(IPC_PRIVATE, IPC_CREAT | 0666);
    struct msgbuf message;
    message.mtype = 1;

    for (int i = 1; i <= 4; i++) {
        sprintf(message.mtext, "Message %d", i);
        msgsnd(msgid, &message, sizeof(message.mtext), 0);
    }
    return 0;
}
```

Receiver Code:

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

struct msgbuf {
    long mtype;
    char mtext[100];
};

int main() {
    int msgid = msgget(IPC_PRIVATE, 0666);
    struct msgbuf message;

    while (msgrcv(msgid, &message, sizeof(message.mtext), 0, IPC_NOWAIT) > 0) {
        printf("Received: %s\n", message.mtext);
    }
    return 0;
}
```

5∗. Write a C code to create a message queue. Write 6 messages of message the type 10, 30, 46, 67, 78, and 88 onto the queue. Create a receiver code to receive the message depending on the **msgtyp** parameter of the **msgrcv** system call as **msgtyp=-10**, **msgtyp=100**, **msgtyp=-46**, **msgtyp=0**, and **msgtyp=88** respectively.

Sender Code:

c Copy code

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

struct msgbuf {
    long mtype;
    char mtext[100];
};

int main() {
    int msgid = msgget(IPC_PRIVATE, IPC_CREAT | 0666);
    if (msgid == -1) {
        perror("msgget failed");
        return 1;
    }

    struct msgbuf message;
    long types[] = {10, 30, 46, 67, 78, 88};
    char *texts[] = {"Message 10", "Message 30", "Message 46", "Message 67", "Message 78",

    for (int i = 0; i < 6; i++) {
        message.mtype = types[i];
        strcpy(message.mtext, texts[i]);
        if (msgsnd(msgid, &message, sizeof(message.mtext), 0) == -1) {
            perror("msgsnd failed");
            return 1;
        }
        printf("Sent: %s\n", message.mtext);
    }

    return 0;
}
```

Receiver Code:

c Copy code

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

struct msgbuf {
    long mtype;
    char mtext[100];
};

int main() {
    int msgid = msgget(IPC_PRIVATE, 0666);
    if (msgid == -1) {
        perror("msgget failed");
        return 1;
    }

    struct msgbuf message;
    long filters[] = {-10, 100, -46, 0, 88};
    int filter_count = sizeof(filters) / sizeof(filters[0]);

    for (int i = 0; i < filter_count; i++) {
        if (msgrcv(msgid, &message, sizeof(message.mtext), filters[i], IPC_NOWAIT) == -1)
            perror("msgrcv failed");
        } else {
            printf("Received with msgtyp %ld: %s\n", filters[i], message.mtext);
        }
    }

    return 0;
}
```

6∗. Write a program to read a string iter and encrypt the string using a cryptographic technique called caeser cipher with a key value of 5. The encryption can be represented using modular arithmetic by first transforming the letters into numbers, according to the scheme, $A = 0$, $B = 1$,..., $Z = 25$. Encryption of a letter say x by a key k can be described mathematically as $E_k(x) = (x + k) \bmod 26$. After encryption write the key value and encrypted message on to the queue. Create a receiver code to get the message and the key value. Decrypt the received message using the reverse process as $D_k(x) = (x-k) \bmod 26$. After decryption display the message on the receiver side.

Example-1

Text : asdzf
key: 3
Cipher: dvgci

Example-2

Text : ATTACKATONCE
Shift: 4

Sender Code:

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

struct msgbuf {
    long mtype;
    char mtext[100];
};

void encrypt(char *str, int key) {
    for (int i = 0; str[i]; i++) {
        str[i] = ((str[i] - 'a' + key) % 26) + 'a';
    }
}

int main() {
    int msgid = msgget(IPC_PRIVATE, IPC_CREAT | 0666);
    struct msgbuf message;
    message.mtype = 1;
    char text[] = "hello";

    encrypt(text, 5);
    sprintf(message.mtext, "%s:%d", text, 5);
    msgsnd(msgid, &message, sizeof(message.mtext), 0);

    return 0;
}
```

Receiver Code:

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

struct msgbuf {
    long mtype;
    char mtext[100];
};

void decrypt(char *str, int key) {
    for (int i = 0; str[i]; i++) {
        str[i] = ((str[i] - 'a' - key + 26) % 26) + 'a';
    }
}

int main() {
    int msgid = msgget(IPC_PRIVATE, 0666);
    struct msgbuf message;

    msgrcv(msgid, &message, sizeof(message.mtext), 0, 0);

    char text[100];
    int key;
    sscanf(message.mtext, "%[^:]:%d", text, &key);
    decrypt(text, key);

    printf("Decrypted Message: %s\n", text);
    return 0;
}
```

Code here Specify: input & output