Politechnika Świętokrzyska
Kielce University of Technology

Operating systems 1

IPC communication – message queues

Tanmay Nandanikar | Lab report 4 | 11/6/20

dr inż. Karol Tomaszewski

INTRODUCTION

This lab primarily focused on Inter-process communication (IPC) using message queues. The Linux message queue API for the C language was introduced in this lab.

1 BRIEF OVERVIEW OF THEORY

1.1 INTER-PROCESS COMMUNICATION IN THE SYSTEM V RELEASE

IPC in the System V release is possible through *message queues, semaphores,* and *shared memory*. It is handled by the Kernel at the request of User programs. There is also an alternative **`POSIX`** compliant implementation for IPC.

1.2 MESSAGE QUEUES

Within the context of this lab, it is possible to access message queues through the appropriate C library functions. In Linux, a message is limited to 8 KB in size, while a message queue is limited to 16 KB. They are more flexible in terms of accessibility compared to named pipes.

1.3 MESSAGE QUEUE API

C programs using message queues need to include the following header files:

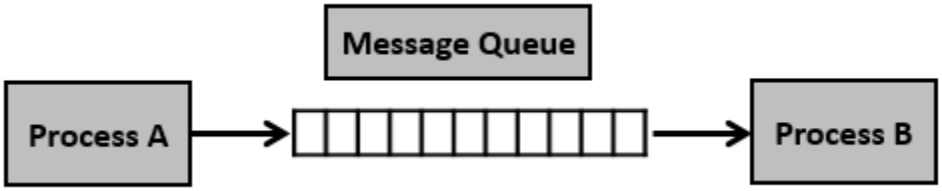
- `sys/types.h` - contains type definitions
- `sys/ipc.h` - contains function declarations and structure definitions common to all ipc mechanisms
- `sys/msg.h` - contains functions and structures designed solely to handle message queues.

As long as the message type is specified, a program can send a message in any user-defined structure.

Generally, the essential functions to use a message queue in C are:

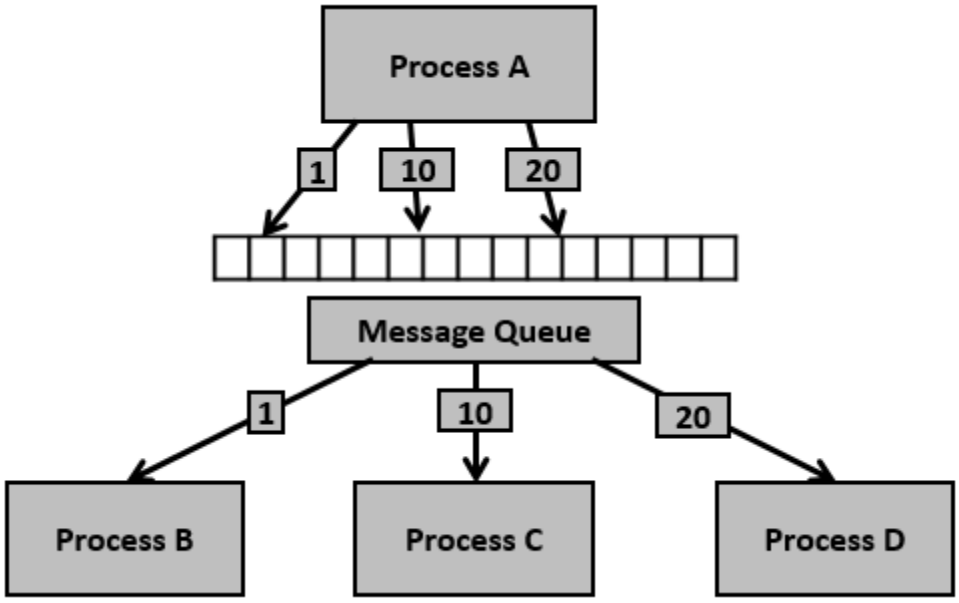
- `ftok()` – to get a key for the message queue
- `msgget()` – create message queue
- `msgsnd()` – send message to the message queue
- `msgrcv()` – receive message from the message queue
- `msgctl()` – perform control operations on the message queue

1.3.1 Communication between 2 processes using a Message Queue



Listing 1[1]: Visualization of writing and reading from shared memory

1.3.2 Writing multiple data packets read by different processes



Listing 2[2]: Writing into the shared memory by one process with different data packets and reading from it by multiple processes

[1] Figure taken from tutorialspoint.com

[2] Figure taken from tutorialspoint.com

2 EXERCISES

2.1 TASK 1

A program that utilizes a private message queue. Source code can be found in the source section (here). Output for this task is shown below:

2.1.1 Output for Task 1

```
sprices@pop-os:~/Documents/operating systems/lab4$ ./a.out
Received message: Operating Systems
```

2.2 TASK 2

Task 1 modified to communicate with a child process. Source code can be found in the source section (here). Output for this task is shown below:

2.2.1 Output for Task 2

```
sprices@pop-os:~/Documents/operating systems/lab4$ ./a.out
Received message: Operating Systems
```

2.3 TASK 3

Output with the NOWAIT flag where program 1 is running, without the NOWAIT flag where program 1 is not running, and output with the NOWAIT flag where program 1 is not running. In part 2, program waits for the message queue, while in 3 it returns an error. Source code can be found in the source section (here). Output for this task is shown below:

2.3.1 Output for Task 3

```
1
sprices@pop-os:~/Documents/operating systems/lab4$ ./ex3_1
Received message: Operating Systems

2
sprices@pop-os:~/Documents/operating systems/lab4$ ./ex3_1
^C

3
sprices@pop-os:~/Documents/operating systems/lab4$ ./ex3_1
msgrcv: No message of desired type
```

2.4 TASK 4

Messages with random type variables, arranged in ascending and descending order by type value. Source code can be found in the source section (here). Output for this task is shown below:

```
sprices@pop-os:~/Documents/operating systems/lab4$ ./ex4_1
Received message type 1: Operating Systems
Received message type 1: Operating Systems



Received message type 2: Operating Systems
Received message type 2: Operating Systems
Received message type 2: Operating Systems


Received message type 3: Operating Systems
Received message type 3: Operating Systems



Received message type 4: Operating Systems
Received message type 4: Operating Systems



Received message type 5: Operating Systems




sprices@pop-os:~/Documents/operating systems/lab4$ ./ex4_1 1 2 3 4 5
Received message type 5: Operating Systems



Received message type 4: Operating Systems
Received message type 4: Operating Systems



Received message type 3: Operating Systems
Received message type 3: Operating Systems



Received message type 2: Operating Systems
Received message type 2: Operating Systems
Received message type 2: Operating Systems


Received message type 1: Operating Systems
Received message type 1: Operating Systems
```

2.5 TASK 5

Ten messages from program 1, printed in ascending and descending order. Source code can be found in the source section (here). Output for this task is shown below

2.5.1 Output for Task 5

```
sprices@pop-os:~/Documents/operating systems/lab4$ ./ex5_1
Received message type 1: Operating Systems
Received message type 2: Operating Systems
Received message type 3: Operating Systems
Received message type 4: Operating Systems
Received message type 5: Operating Systems
Received message type 6: Operating Systems
Received message type 7: Operating Systems
Received message type 8: Operating Systems
Received message type 9: Operating Systems
Received message type 10: Operating Systems
sprices@pop-os:~/Documents/operating systems/lab4$ ./ex5_1 1 2 3 4 5
Received message type 10: Operating Systems
Received message type 9: Operating Systems
Received message type 8: Operating Systems
Received message type 7: Operating Systems
Received message type 6: Operating Systems
Received message type 5: Operating Systems
Received message type 4: Operating Systems
Received message type 3: Operating Systems
Received message type 2: Operating Systems
Received message type 1: Operating Systems
```

2.6 TASK 6

Two-way IPC between 3 different programs over a shared queue with custom message reception. Source code can be found in the source section (here). Output for this task is shown below:

2.6.1 Output for Task 6

```
sprices@pop-os:~/Documents/operating systems/lab4$ ./ex6_1
P2 received message 1: Operating Systems
P2 received message 2: Operating Systems

sprices@pop-os:~/Documents/operating systems/lab4$

sprices@pop-os:~/Documents/operating systems/lab4$ ./ex6_2
P3 received message 1: Operating Systems
P3 received message 2: Operating Systems

sprices@pop-os:~/Documents/operating systems/lab4$

sprices@pop-os:~/Documents/operating systems/lab4$ ./a.out
```

```
P1 received message 1: Operating Systems
P1 received message 2: Operating Systems
sprices@pop-os:~/Documents/operating systems/lab4$
```

2.7 TASK 7

I tried sending a pointer through the message queue for this task but a segmentation fault occurred. I'm sure there are better ways to go about this task that I could not think of. If this task only requires a message with the length sent then I could do that, but I could not think of how to send the message using a dynamic variable. Source code can be found in the source section (here). Output for this task is shown below:

2.7.1 Output for Task 7

```
sprices@pop-os:~/Documents/operating systems/lab4/dump$ ./a.out
Segmentation fault (core dumped)
```

2.8 TASK 8

Terminal chat program using mknod(). Source code can be found in the source section (here). Output for this task is shown below:

2.8.1 Output for Task 8

```
sprices@pop-os:~/Documents/operating systems/lab4$ ./ex8_2
1 + 2 = 3
3 + 4 = 7
5 + 6 = 11
7 + 8 = 15
9 + 10 = 19
11 + 12 = 23
13 + 14 = 27
15 + 16 = 31
17 + 18 = 35
19 + 20 = 39
21 + 22 = 43
23 + 24 = 47
25 + 26 = 51
27 + 28 = 55
29 + 30 = 59
```

3 CONCLUSION

During this lab, I learnt about Inter-process communication using message queues. I learnt how to use message queue functions with and without various flags such as the `IPC_NOWAIT` flag and the `IPC_PRIVATE` flag.

4 SOURCE

4.1 SOURCE CODE FOR EXERCISE ONE

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

#define TEXT_LENGTH 50

struct msgbuf {
long type;
char mtext[TEXT_LENGTH];
};

void send_message(int mqid, char message[])
{
struct msgbuf buffer;

buffer.type = 1;
memset(buffer.mtext,0,sizeof(buffer.mtext));
strncpy(buffer.mtext,message,TEXT_LENGTH-1);

if(msgsnd(mqid,&buffer,sizeof(buffer.mtext),0)<0)
perror("msgsnd");
}

void receive_message(int mqid)
{
struct msgbuf buffer;

if(msgrcv(mqid,&buffer,sizeof(buffer.mtext),1,0)<0)
perror("msgrcv");
else
printf("Received message: %s\n",buffer.mtext);
}


int remove_queue( int qid )
{
        if( msgctl( qid, IPC_RMID, 0) == -1)
        {
```

```
                return(-1);
        }

        return(0);
}


int main(void)
{
int id = msgget(IPC_PRIVATE, 0600);
if(id<0)
perror("msgget");

send_message(id,"Operating Systems");
receive_message(id);
remove_queue(id);

 return 0;
}
```

4.2 SOURCE CODE FOR EXERCISE TWO

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

#define TEXT_LENGTH 50

struct msgbuf {
long type;
char mtext[TEXT_LENGTH];
};

void send_message(int mqid, char message[])
{
struct msgbuf buffer;

buffer.type = 1;
memset(buffer.mtext,0,sizeof(buffer.mtext));
strncpy(buffer.mtext,message,TEXT_LENGTH-1);

if(msgsnd(mqid,&buffer,sizeof(buffer.mtext),0)<0)
perror("msgsnd");
}

void receive_message(int mqid)
{
struct msgbuf buffer;
```

```
if(msgrcv(mqid,&buffer,sizeof(buffer.mtext),1,0)<0)
perror("msgrcv");
else
printf("Received message: %s\n",buffer.mtext);
}


int remove_queue( int qid )
{
        if( msgctl( qid, IPC_RMID, 0) == -1)
        {
                return(-1);
        }

        return(0);
}


int main(void)
{
int id = msgget(IPC_PRIVATE, 0600);
int a=fork();

if(id<0)
        perror("msgget");


if(a!=0){
receive_message(id);
}
else{
send_message(id,"Operating Systems");
}

remove_queue(id);

 return 0;
}
```

4.3 SOURCE CODE FOR EXERCISE THREE

4.3.1 Part 1

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

#define TEXT_LENGTH 50

struct msgbuf {
long type;
char mtext[TEXT_LENGTH];
```

```
};

void send_message(int mqid, char message[])
{
struct msgbuf buffer;

buffer.type = 1;
memset(buffer.mtext,0,sizeof(buffer.mtext));
strncpy(buffer.mtext,message,TEXT_LENGTH-1);

if(msgsnd(mqid,&buffer,sizeof(buffer.mtext),0)<0)
perror("msgsnd");
}

void receive_message(int mqid)
{
struct msgbuf buffer;

if(msgrcv(mqid,&buffer,sizeof(buffer.mtext),1,0)<0)
perror("msgrcv");
else
printf("Received message: %s\n",buffer.mtext);
}

int remove_queue( int qid )
{
        if( msgctl( qid, IPC_RMID, 0) == -1)
        {
                return(-1);
        }

        return(0);
}


int main(void)
{
int key = ftok("/tmp",8);

if(key<0)
       perror("ftok");

int id = msgget(key, 0600 |IPC_CREAT|IPC_NOWAIT);

if(id<0)
       perror("msgget");

send_message(id,"Operating Systems");
//receive_message(id);
sleep(50);
remove_queue( id );

return 0;
}
```

4.3.2 Part 2

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

#define TEXT_LENGTH 50

struct msgbuf {
long type;
char mtext[TEXT_LENGTH];
};

void send_message(int mqid, char message[])
{
struct msgbuf buffer;

buffer.type = 1;
memset(buffer.mtext,0,sizeof(buffer.mtext));
strncpy(buffer.mtext,message,TEXT_LENGTH-1);

if(msgsnd(mqid,&buffer,sizeof(buffer.mtext),0)<0)
perror("msgsnd");
}

void receive_message(int mqid)
{
struct msgbuf buffer;

if(msgrcv(mqid,&buffer,sizeof(buffer.mtext),1,0 |IPC_NOWAIT)<0)
perror("msgrcv");
else
printf("Received message: %s\n",buffer.mtext);
}

int remove_queue( int qid )
{
        if( msgctl( qid, IPC_RMID, 0) == -1)
        {
                return(-1);
        }

        return(0);
}


int main(void)
{
int key = ftok("/tmp",8);

if(key<0)
       perror("ftok");

int id = msgget(key, 0600 |IPC_CREAT|IPC_EXCL);
```

```
if(id<0)
        perror("msgget");

//send_message(id,"Operating Systems");
receive_message(id);
remove_queue( id );

return 0;
}
```

4.4 SOURCE CODE FOR EXERCISE FOUR

4.4.1 Part 1

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

#define TEXT_LENGTH 50

struct msgbuf {
long type;
char mtext[TEXT_LENGTH];
};

int send_message(int mqid, char message[])
{
struct msgbuf buffer;
int r=rand() % 5 + 1;
buffer.type = r;
memset(buffer.mtext,0,sizeof(buffer.mtext));
strncpy(buffer.mtext,message,TEXT_LENGTH-1);

if(msgsnd(mqid,&buffer,sizeof(buffer.mtext),0)<0)
    perror("msgsnd");

return r;
}


int remove_queue( int qid )
{
        if( msgctl( qid, IPC_RMID, 0) == -1)
        {
                return(-1);
        }
```

```
        return(0);
}


int main(void)
{
int key = ftok("/tmp",8);

if(key<0)
    perror("ftok");

int id = msgget(key, 0600 |IPC_CREAT|IPC_NOWAIT);

if(id<0)
    perror("msgget");

int i,r;
for(i=0;i<10;i++){
r=send_message(id,"Operating Systems");
printf("%d ",r);
}
printf("\n");
//receive_message(id);
sleep(5);
remove_queue( id );

return 0;
}
```

4.4.2 Part 2

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

#define TEXT_LENGTH 50

struct msgbuf {
long type;
char mtext[TEXT_LENGTH];
};

void send_message(int mqid, char message[])
{
struct msgbuf buffer;

buffer.type = 1;
memset(buffer.mtext,0,sizeof(buffer.mtext));
strncpy(buffer.mtext,message,TEXT_LENGTH-1);

if(msgsnd(mqid,&buffer,sizeof(buffer.mtext),0)<0)
perror("msgsnd");
}
```

```c
void receive_message(int mqid)
{
struct msgbuf buffer;
int i;

for(i=1;i<6;i++){
if(msgrcv(mqid,&buffer,sizeof(buffer.mtext),i,0 |IPC_NOWAIT)<0)
printf("\n");
else
printf("Received message type %d: %s\n",i,buffer.mtext);

if(msgrcv(mqid,&buffer,sizeof(buffer.mtext),i,0 |IPC_NOWAIT)<0)
printf("\n");
else
printf("Received message type %d: %s\n",i,buffer.mtext);

if(msgrcv(mqid,&buffer,sizeof(buffer.mtext),i,0 |IPC_NOWAIT)<0)
printf("\n");
else
printf("Received message type %d: %s\n",i,buffer.mtext);

if(msgrcv(mqid,&buffer,sizeof(buffer.mtext),i,0 |IPC_NOWAIT)<0)
printf("\n");
else
printf("Received message type %d: %s\n",i,buffer.mtext);

if(msgrcv(mqid,&buffer,sizeof(buffer.mtext),i,0 |IPC_NOWAIT)<0)
printf("\n");
else
printf("Received message type %d: %s\n",i,buffer.mtext);
}
}

void receive_message_reverse(int mqid)
{
struct msgbuf buffer;
int i;

for(i=5;i>0;i--){
if(msgrcv(mqid,&buffer,sizeof(buffer.mtext),i,0 |IPC_NOWAIT)<0)
printf("\n");
else
printf("Received message type %d: %s\n",i,buffer.mtext);

if(msgrcv(mqid,&buffer,sizeof(buffer.mtext),i,0 |IPC_NOWAIT)<0)
printf("\n");
else
printf("Received message type %d: %s\n",i,buffer.mtext);

if(msgrcv(mqid,&buffer,sizeof(buffer.mtext),i,0 |IPC_NOWAIT)<0)
printf("\n");
else
printf("Received message type %d: %s\n",i,buffer.mtext);
```

```
if(msgrcv(mqid,&buffer,sizeof(buffer.mtext),i,0 |IPC_NOWAIT)<0)
printf("\n");
else
printf("Received message type %d: %s\n",i,buffer.mtext);

if(msgrcv(mqid,&buffer,sizeof(buffer.mtext),i,0 |IPC_NOWAIT)<0)
printf("\n");
else
printf("Received message type %d: %s\n",i,buffer.mtext);
}
}

int remove_queue( int qid )
{
        if( msgctl( qid, IPC_RMID, 0) == -1)
        {
                return(-1);
        }

        return(0);
}


int main(int argc, char *argv[])
{
int key = ftok("/tmp",8);

if(key<0)
        perror("ftok");

int id = msgget(key, 0600 |IPC_CREAT);

if(id<0)
        perror("msgget");

//send_message(id,"Operating Systems");
if(argc<3){
receive_message(id);
}
else{
receive_message_reverse(id);
}

remove_queue( id );

return 0;
}
```

4.5 SOURCE CODE FOR EXERCISE FIVE

4.5.1 Part 1

```c
#include<stdio.h>
#include<stdlib.h>
#include<sys/ipc.h>
#include<sys/msg.h>
#include<sys/types.h>
#include<unistd.h>
#include<string.h>
#include<time.h>

#define TEXT_LENGTH 50

struct msgbuf {
long type;
char mtext[TEXT_LENGTH];
};

int send_messages(int mqid, char message[])
{
struct msgbuf buffer;
int i;
for(i=1;i<11;i++){
buffer.type = i;
memset(buffer.mtext,0,sizeof(buffer.mtext));
strncpy(buffer.mtext,message,TEXT_LENGTH-1);

if(msgsnd(mqid,&buffer,sizeof(buffer.mtext),0)<0)
    perror("msgsnd");
}
return 0;
}


int remove_queue( int qid )
{
        if( msgctl( qid, IPC_RMID, 0) == -1)
        {
                return(-1);
        }

        return(0);
}


int main(void)
{
int key = ftok("/tmp",8);

if(key<0)
    perror("ftok");

int id = msgget(key, 0600 |IPC_CREAT|IPC_NOWAIT);

if(id<0)
    perror("msgget");
```

```
int i,r;

send_messages(id,"Operating Systems");

printf("\n");
//receive_message(id);
sleep(25);
remove_queue( id );

return 0;
}
```

4.5.2 Part 2

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

#define TEXT_LENGTH 50

struct msgbuf {
long type;
char mtext[TEXT_LENGTH];
};


void receive_message(int mqid)
{
struct msgbuf buffer;
int i;

for(i=1;i<11;i++){
if(msgrcv(mqid,&buffer,sizeof(buffer.mtext),i,0 |IPC_NOWAIT)<0)
perror("msgrcv");
else
printf("Received message type %d: %s\n",i,buffer.mtext);
}
}

void receive_message_reverse(int mqid)
{
struct msgbuf buffer;
int i;

for(i=10;i>0;i--){
if(msgrcv(mqid,&buffer,sizeof(buffer.mtext),i,0 |IPC_NOWAIT)<0)
perror("msgrcv");
else
printf("Received message type %d: %s\n",i,buffer.mtext);
}
}
```

```
int remove_queue( int qid )
{
        if( msgctl( qid, IPC_RMID, 0) == -1)
        {
                return(-1);
        }

        return(0);
}


int main(int argc, char *argv[])
{
int key = ftok("/tmp",8);

if(key<0)
        perror("ftok");

int id = msgget(key, 0600 |IPC_CREAT);

if(id<0)
        perror("msgget");

//send_message(id,"Operating Systems");
if(argc<3){
receive_message(id);
}
else{
receive_message_reverse(id);
}

remove_queue( id );

return 0;
}
```

4.6 Source code for Exercise Six

4.6.1 Part 1

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

#define TEXT_LENGTH 50

struct msgbuf {
long type;
char mtext[TEXT_LENGTH];
```

```
};

int send_messages(int mqid, char message[])
{
struct msgbuf buffer;

buffer.type = 2;
memset(buffer.mtext,0,sizeof(buffer.mtext));
strncpy(buffer.mtext,message,TEXT_LENGTH-1);

if(msgsnd(mqid,&buffer,sizeof(buffer.mtext),0)<0)
    perror("msgsnd");

buffer.type = 3;
memset(buffer.mtext,0,sizeof(buffer.mtext));
strncpy(buffer.mtext,message,TEXT_LENGTH-1);

if(msgsnd(mqid,&buffer,sizeof(buffer.mtext),0)<0)
    perror("msgsnd");

return 0;
}

void receive_messages(int mqid)
{
struct msgbuf buffer;

if(msgrcv(mqid,&buffer,sizeof(buffer.mtext),1,0)<0)
perror("msgrcv");
else
printf("P1 received message 1: %s\n",buffer.mtext);

if(msgrcv(mqid,&buffer,sizeof(buffer.mtext),1,0)<0)
perror("msgrcv");
else
printf("P1 received message 2: %s\n",buffer.mtext);
}

int remove_queue( int qid )
{
        if( msgctl( qid, IPC_RMID, 0) == -1)
        {
                return(-1);
        }

        return(0);
}


int main(void)
{
int key = ftok("/tmp",8);

if(key<0)
    perror("ftok");
```

```
int id = msgget(key, 0600 |IPC_CREAT|IPC_NOWAIT);

if(id<0)
    perror("msgget");

int i,r;

send_messages(id,"Operating Systems");
printf("\n");
sleep(25);

receive_messages(id);

remove_queue( id );

return 0;
}
```

4.6.2 Part 2

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

#define TEXT_LENGTH 50

struct msgbuf {
long type;
char mtext[TEXT_LENGTH];
};

int send_messages(int mqid, char message[])
{
struct msgbuf buffer;

buffer.type = 1;
memset(buffer.mtext,0,sizeof(buffer.mtext));
strncpy(buffer.mtext,message,TEXT_LENGTH-1);

if(msgsnd(mqid,&buffer,sizeof(buffer.mtext),0)<0)
    perror("msgsnd");

buffer.type = 3;
memset(buffer.mtext,0,sizeof(buffer.mtext));
strncpy(buffer.mtext,message,TEXT_LENGTH-1);

if(msgsnd(mqid,&buffer,sizeof(buffer.mtext),0)<0)
```

```
    perror("msgsnd");

return 0;
}

void receive_messages(int mqid)
{
struct msgbuf buffer;

if(msgrcv(mqid,&buffer,sizeof(buffer.mtext),2,0)<0)
perror("msgrcv");
else
printf("P2 received message 1: %s\n",buffer.mtext);

if(msgrcv(mqid,&buffer,sizeof(buffer.mtext),2,0)<0)
perror("msgrcv");
else
printf("P2 received message 2: %s\n",buffer.mtext);
}

int remove_queue( int qid )
{
        if( msgctl( qid, IPC_RMID, 0) == -1)
        {
                return(-1);
        }

        return(0);
}


int main(void)
{
int key = ftok("/tmp",8);

if(key<0)
    perror("ftok");

int id = msgget(key, 0600 |IPC_CREAT|IPC_NOWAIT);

if(id<0)
    perror("msgget");

int i,r;
send_messages(id,"Operating Systems");
receive_messages(id);
printf("\n");

return 0;
}
```

4.6.3 Part 3

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

#define TEXT_LENGTH 50

struct msgbuf {
long type;
char mtext[TEXT_LENGTH];
};

int send_messages(int mqid, char message[])
{
struct msgbuf buffer;

buffer.type = 1;
memset(buffer.mtext,0,sizeof(buffer.mtext));
strncpy(buffer.mtext,message,TEXT_LENGTH-1);

if(msgsnd(mqid,&buffer,sizeof(buffer.mtext),0)<0)
    perror("msgsnd");

buffer.type = 2;
memset(buffer.mtext,0,sizeof(buffer.mtext));
strncpy(buffer.mtext,message,TEXT_LENGTH-1);

if(msgsnd(mqid,&buffer,sizeof(buffer.mtext),0)<0)
    perror("msgsnd");

return 0;
}

void receive_messages(int mqid)
{
struct msgbuf buffer;

if(msgrcv(mqid,&buffer,sizeof(buffer.mtext),3,0)<0)
perror("msgrcv");
else
printf("P3 received message 1: %s\n",buffer.mtext);

if(msgrcv(mqid,&buffer,sizeof(buffer.mtext),3,0)<0)
perror("msgrcv");
else
printf("P3 received message 2: %s\n",buffer.mtext);
}

int remove_queue( int qid )
{
```

```
        if( msgctl( qid, IPC_RMID, 0) == -1)
        {
                return(-1);
        }

        return(0);
}


int main(void)
{
int key = ftok("/tmp",8);

if(key<0)
    perror("ftok");

int id = msgget(key, 0600 |IPC_CREAT|IPC_NOWAIT);

if(id<0)
    perror("msgget");

int i,r;
send_messages(id,"Operating Systems");
receive_messages(id);
printf("\n");

return 0;
}
```

4.7 SOURCE CODE FOR EXERCISE SEVEN

4.7.1 Part 1

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


struct msgbuf {
long type;
int length;
};

struct msgbuf1 {
long type;
char *mtext;
};

void send_message(int mqid, char message[],int l)
```

```
{
struct msgbuf buffer;

buffer.type = 1;
memset(&buffer.length,0,sizeof(buffer.length));
buffer.length=l;

if(msgsnd(mqid,&buffer,sizeof(buffer.length),0)<0)
perror("msgsnd");



struct msgbuf1 buffer1;

buffer1.type = 2;
memset(&buffer1.mtext,0,sizeof(char[l]));
strncpy(buffer1.mtext,message,l-1);

if(msgsnd(mqid,&buffer1,sizeof(char[l]),0)<0)
perror("msgsnd");
}

int remove_queue( int qid )
{
        if( msgctl( qid, IPC_RMID, 0) == -1)
        {
                return(-1);
        }

        return(0);
}


int main(void)
{
int key = ftok("/tmp",8);

if(key<0)
       perror("ftok");

int id = msgget(key, 0600 |IPC_CREAT|IPC_NOWAIT);

if(id<0)
       perror("msgget");

char text[50]="Operating Systems";
send_message(id,text, strlen(text));
//receive_message(id);
sleep(30);
remove_queue( id );

return 0;
}
```

4.7.2 Part 2

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

#define TEXT_LENGTH 128

struct msgbuf {
long type;
int length;
};

struct msgbuf1 {
long type;
char *mtext;
};


void receive_message(int mqid)
{
struct msgbuf buffer;

if(msgrcv(mqid,&buffer,sizeof(buffer.length),1,0)<0)
perror("msgrcv");
else{
int temp=buffer.length;
printf("%d", temp);
}

char temp1[buffer.length];

struct msgbuf1 buffer1;
if(msgrcv(mqid,&buffer1,sizeof(temp1),2,0)<0)
perror("msgrcv");
else{
printf("Recieved %s\n", buffer1.mtext);
}
}

int remove_queue( int qid )
{
        if( msgctl( qid, IPC_RMID, 0) == -1)
        {
                return(-1);
        }

        return(0);
}


int main(void)
{
int key = ftok("/tmp",8);
```

```
if(key<0)
        perror("ftok");

int id = msgget(key, 0600 |IPC_CREAT|IPC_EXCL);

if(id<0)
        perror("msgget");

//send_message(id,"Operating Systems");
receive_message(id);
remove_queue( id );

return 0;
}
```

4.8 SOURCE CODE FOR EXERCISE EIGHT

4.8.1 Part 1

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

#define TEXT_LENGTH 50

struct msgbuf {
long type;
int num;
};


int send_messages(int mqid, int n)
{
struct msgbuf buffer;

buffer.type = 1;
memset(&buffer.num,0,sizeof(buffer.num));
buffer.num=n;

if(msgsnd(mqid,&buffer,sizeof(buffer.num),0)<0)
    perror("msgsnd");

return 0;
}


int remove_queue( int qid )
```

```
{
        if( msgctl( qid, IPC_RMID, 0) == -1)
        {
                return(-1);
        }

        return(0);
}


int main(void)
{
int key = ftok("/tmp",8);

if(key<0)
    perror("ftok");

int id = msgget(key, 0600 |IPC_CREAT|IPC_NOWAIT);

if(id<0)
    perror("msgget");

int i,r;
for(i=1;i<30;i+=2){
send_messages(id,i);
}
printf("\n");
sleep(35);


remove_queue( id );

return 0;
}
```

4.8.2 Part 2

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

#define TEXT_LENGTH 50

struct msgbuf {
long type;
int num;
};


int send_messages(int mqid, int n)
```

```
{
struct msgbuf buffer;

buffer.type = 2;
memset(&buffer.num,0,sizeof(buffer.num));
buffer.num=n;

if(msgsnd(mqid,&buffer,sizeof(buffer.num),0)<0)
    perror("msgsnd");

return 0;
}


int main(void)
{
int key = ftok("/tmp",8);

if(key<0)
    perror("ftok");

int id = msgget(key, 0600 |IPC_CREAT|IPC_NOWAIT);

if(id<0)
    perror("msgget");

int i,r;
for(i=2;i<=30;i+=2){
send_messages(id,i);
}
printf("\n");

sleep(35);

return 0;
}
```

4.8.3 Part 3

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


struct msgbuf {
long type;
int num;
};

void receive_messages(int mqid)
{
```

```
struct msgbuf buffer;
int temp,temp1;
int i;
for(i=0;i<15;i++){
if((msgrcv(mqid,&buffer,sizeof(buffer.num),1,0)<0))
perror("msgrcv");
else
temp=buffer.num;

if(msgrcv(mqid,&buffer,sizeof(buffer.num),2,0)<0)
perror("msgrcv");
else
temp1=buffer.num;

printf("%d + %d = %d\n",temp,temp1,temp+temp1);
}
}

int remove_queue( int qid )
{
        if( msgctl( qid, IPC_RMID, 0) == -1)
        {
                return(-1);
        }

        return(0);
}


int main(void)
{
int key = ftok("/tmp",8);

if(key<0)
    perror("ftok");

int id = msgget(key, 0600 |IPC_CREAT|IPC_NOWAIT);

if(id<0)
    perror("msgget");

receive_messages(id);

printf("\n");
remove_queue(id);

return 0;
}
```