Politechnika Świętokrzyska
Kielce University of Technology

Operating systems 1

IPC communication – semaphores

Tanmay Nandanikar | Lab report 5 | 11/13/20

dr inż. Karol Tomaszewski

INTRODUCTION

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

1 BRIEF OVERVIEW OF THEORY

1.1 SEMAPHORES

Semaphores are abstract data types that have two indivisible operations defined, namely, `p – wait` and `v – signal`. A binary semaphore is the simplest type of semaphore and which can only have two values, 0 and 1, available through p and v operations. Operation p reduces the value off a semaphore, and suspends the process if the value is less than 1. Operation p resumes a suspended process if the semaphore value is 0, or increases the value by 1 if the semaphore is not 0. Semaphores can only be accessed using the appropriate system calls.

1.2 SEMAPHORE API

C programs using semaphores 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/sem.h` - contains functions and structures designed solely to handle semaphores.

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

- `semget()` – creates a semaphore set and returns its id or returns the id of an existing one, accepts a key generated using `ftok()` or the `IPC_PRIVATE` constant, the number of semaphores in the set, and flags related to the function as arguments
- `semop()` – perform atomic operations on a semaphore
- `semctl()` – used to control the semaphore set

2 EXERCISES

2.1 TASK 1

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

```
Parent Process
Parent Process
Parent Process
Parent Process
Parent Process
Parent Process
Parent Process
Parent Process
Parent Process
Parent Process
Parent Process
Parent Process
Parent Process
Parent Process
Parent Process
Parent Process
Parent Process
Parent Process
Parent Process
Parent Process
Parent Process
Parent Process
Parent Process
Parent Process
Parent Process
Child Process
Child Process
Child Process
Child Process
Child Process
Child Process
Child Process
Child Process
Child Process
Child Process
Child Process
Child Process
Child Process
Child Process
Child Process
Child Process
Child Process
Child Process
Child Process
Child Process
Child Process
Child Process
Child Process
Child Process
```

```
Child Process
Child Process
Child Process
Child Process
```

Output for task 1

2.2 TASK 2

Task 1 modified to print messages alternatively instead of all at once. Source code can be found in the source section (here). Output for this task is shown below:

```
Child Process
Parent Process
Child Process
Parent Process
Child Process
Parent Process
Child Process
Parent Process
Child Process
Parent Process
.
.
.
```

Output for task 2

2.3 TASK 3

Utilization of child processes alongside semaphores. Source code can be found in the source section (here). Output for this task is shown below:

```
VAL 0 = 1
VAL 1 = 1
VAL 2 = 1
VAL 3 = 1
VAL 4 = 1
VAL 5 = 1
VAL 6 = 1
VAL 7 = 1
VAL 8 = 1
VAL 9 = 1
OP 0, 0
OP 1, 0
OP 2, 0
OP 3, 0
OP 4, 0
OP 5, 0
OP 6, 0
OP 7, 0
OP 8, 0
OP 9, 0
```

Output for task 3

2.4 TASK 4

Processes increasing value of a semaphore. Source code can be found in the source section (here). Output for this task is shown below:

```
VAL = 0
VAL = 1
VAL = 2
VAL = 3
VAL = 4
VAL = 5
```

Output for task 4

2.5 TASK 5

Synchronization of named pipes. Source code can be found in the source section (here). Output for this task is shown below

```
Terminal 1
P1 wrote 3
P1 wrote 5
P1 wrote 7
P1 wrote 9
P1 wrote 11
P1 wrote 13
P1 wrote 15
P1 wrote 17
P1 wrote 19
P1 wrote 21

Terminal 2
Op success
User1: 3
Op success
User1: 5
Op success
User1: 7
Op success
User1: 9
Op success
User1: 11
Op success
User1: 13
Op success
User1: 15
Op success
User1: 17
Op success
User1: 19
Op success
```

```
User1: 21
```

Output for task 5

2.6 Task 6

Demonstration of a deadlock (original code from the internet, which I modified to create a deadlock). Source code can be found in the source section (here). Output for this task is shown below:

```
The Dining-Philosophers Problem
Philosopher 0 at the table
Philosopher 2 at the table
Philosopher 3 at the table
Philosopher 4 at the table
Philosopher 1 at the table
 0: Philosopher 3 is thinking ...
 0: Philosopher 4 is thinking ...
 0: Philosopher 0 is thinking ...
 0: Philosopher 1 is thinking ...
 0: Philosopher 2 is thinking ...
^C
```

Output for task 6

2.7 Task 7

`IPC_UNDO` flag demonstration. Without the flag, 2.2 and 1.1 would be equal. Source code can be found in the source section (here). Output for this task is shown below:

```
Val 2.1 = 1
Child Process
Child Process
Child Process
Child Process
Child Process
Val 2.2 = 0
Val 1.1 = 1
Parent Process
Parent Process
Parent Process
Parent Process
Parent Process
Val 1.2 = 1
```

Output for task 7

2.8 Task 8

I encountered an error during this task. Source code can be found in the source section (here). Output for this task is shown below:

```
 semctl: Numerical result out of range
```

Output for task 8

2.9 TASK 9

Message queue synchronization using semaphores. Source code can be found in the source section (here). Output for this task is shown below:

```
Received message: Operating Systems
```

Output for task 9

2.10 TASK 10

Message queue synchronization using semaphores. Source code can be found in the source section (here). Output for this task is shown below:

```
Got number from child process: 1
Got number from child process: 2
Got number from child process: 3
Got number from child process: 4
Got number from child process: 5
Got number from child process: 6
Got number from child process: 7
Got number from child process: 8
Got number from child process: 9
Got number from child process: 10
END
```

Output for task 10

3 CONCLUSION

During this lab, I learnt about synchronization between processes using semaphores to implement code sequentially. I learnt how synchronize various IPC communication methods discussed in the precious labs such as message queues suing semaphores.

4 SOURCE

4.1 SOURCE CODE FOR EXERCISE ONE

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

```c
#include <sys/sem.h>


union semun {
    int val;
    struct semid_ds *buf;
    unsigned short  *array;
};

struct sembuf p = { 0, -1, SEM_UNDO};
struct sembuf v = { 0, +1, SEM_UNDO};

int main()
{
    int id = semget(IPC_PRIVATE, 1, 0666 | IPC_CREAT);
    if(id < 0)
    {
        perror("semget"); exit(11);
    }
    union semun u;
    u.val = 1;
    if(semctl(id, 0, SETVAL, u) < 0)
    {
        perror("semctl"); exit(12);
    }
    int pid;
    pid =  fork();
    if(pid < 0)
    {
        perror("fork"); exit(1);
    }
    else if(pid)
    {
        sleep(1);
        for(int i = 0; i < 25; ++i)
        {
            if(semop(id, &p, 1) < 0)
            {
                perror("semop p"); exit(13);
            }
            printf("Child Process\n");
            if(semop(id, &v, 1) < 0)
            {
                perror("semop p"); exit(14);
            }
        }
    }
    else
    {
        for(int i = 0; i < 25; ++i)
        {
            if(semop(id, &p, 1) < 0)
            {
                perror("semop p"); exit(15);
            }
```

```
            printf("Parent Process\n");
            if(semop(id, &v, 1) < 0)
            {
                perror("semop p"); exit(16);
            }
        }
    }
}
```

4.2 SOURCE CODE FOR EXERCISE TWO

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


union semun {
    int val;
    struct semid_ds *buf;
    unsigned short  *array;
};

struct sembuf p = { 0, -1, SEM_UNDO};
struct sembuf v = { 0, +1, SEM_UNDO};

int main()
{
    int id = semget(IPC_PRIVATE, 1, 0666 | IPC_CREAT);
    if(id < 0)
    {
        perror("semget"); exit(11);
    }
    union semun u;
    u.val = 1;
    if(semctl(id, 0, SETVAL, u) < 0)
    {
        perror("semctl"); exit(12);
    }
    int pid;
    pid =  fork();
    if(pid < 0)
    {
        perror("fork"); exit(1);
    }
    else if(pid)
    {
        for(int i = 0; i < 25; ++i)
        {
            if(semop(id, &p, 1) < 0)
            {
                perror("semop p"); exit(13);
```

```
            }
            printf("Child Process\n");
            if(semop(id, &v, 1) < 0)
            {
                perror("semop p"); exit(14);
            }
        }
    }
    else
    {
        for(int i = 0; i < 25; ++i)
        {
            if(semop(id, &p, 1) < 0)
            {
                perror("semop p"); exit(15);
            }
            printf("Parent Process\n");
            if(semop(id, &v, 1) < 0)
            {
                perror("semop p"); exit(16);
            }
        }
    }
    sleep(1);
}
```

4.3 SOURCE CODE FOR EXERCISE THREE

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/sem.h>


union semun {
    int val;
    struct semid_ds *buf;
    unsigned short  *array;
};

struct sembuf p = { 0, -1, SEM_UNDO};
struct sembuf p1 = { 1, -1, SEM_UNDO};
struct sembuf p2 = { 2, -1, SEM_UNDO};
struct sembuf p3 = { 3, -1, SEM_UNDO};
struct sembuf p4 = { 4, -1, SEM_UNDO};
struct sembuf p5 = { 5, -1, SEM_UNDO};
struct sembuf p6 = { 6, -1, SEM_UNDO};
struct sembuf p7 = { 7, -1, SEM_UNDO};
struct sembuf p8 = { 8, -1, SEM_UNDO};
struct sembuf p9 = { 9, -1, SEM_UNDO};

int main()
{
    int id = semget(IPC_PRIVATE, 10, 0666 | IPC_CREAT);
```

```
    int i;
    if(id < 0)
    {
        perror("semget"); exit(11);
    }
    union semun u;
    u.val = 1;
    for(i=0;i<10;i++) {
        if (semctl(id, i, SETVAL, u) < 0) {
            perror("semctl");
            exit(12);
        }
    }
    for(i=0;i<10;i++) {
        printf("VAL %d = %d\n",i, semctl(id, i, GETVAL, u));
    }
    if(fork() < 0)
    {
        perror("fork"); exit(1);
    }
    sleep(1);
    if(fork() < 0)
    {
        perror("fork"); exit(1);
    }
    sleep(1);
    if(fork() < 0)
    {
        perror("fork"); exit(1);
    }
    sleep(1);
    if(fork() < 0)
    {
        perror("fork"); exit(1);
    }
    sleep(1);
    if(fork() < 0)
    {
        perror("fork"); exit(1);
    }
    sleep(1);
    if(getppid()) {
        sleep(1);
        for (i = 0; i < 10; i++) {
            if (semctl(id, i, GETVAL, u) == 0) {
                continue;
            }
            if (semctl(id, i, GETVAL, u) == 1) {
                if(i==0){
                    if (semop(id, &p, 1) < 0) {
                        perror("semop p");
                        exit(13);
                    }}
                if (i==1){
                    if (semop(id, &p1, 1) < 0) {
```

```
                        perror("semop p");
                        exit(13);
                    }}
                if (i==2){
                    if (semop(id, &p2, 1) < 0) {
                        perror("semop p");
                        exit(13);}}
                if (i==3){
                    if (semop(id, &p3, 1) < 0) {
                        perror("semop p");
                        exit(13);
                    }}
                if (i==4){
                    if (semop(id, &p4, 1) < 0) {
                        perror("semop p");
                        exit(13);
                    }}
                if (i==5){
                    if (semop(id, &p5, 1) < 0) {
                        perror("semop p");
                        exit(13);
                    }}
                if (i==6){
                    if (semop(id, &p6, 1) < 0) {
                        perror("semop p");
                        exit(13);
                    }}
                if (i==7){
                    if (semop(id, &p7, 1) < 0) {
                        perror("semop p");
                        exit(13);
                    }}
                if (i==8){
                    if (semop(id, &p8, 1) < 0) {
                        perror("semop p");
                        exit(13);
                    }}
                if (i==9){
                    if (semop(id, &p9, 1) < 0) {
                        perror("semop p");
                        exit(13);
                    }}

                printf("OP %d, %d\n", i, semctl(id, i, GETVAL, u));
                break;
            }
        }
    }
    sleep(1);
}
```

4.4 SOURCE CODE FOR EXERCISE FOUR

```
#include <stdio.h>
#include <stdlib.h>
```

```c
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/sem.h>


union semun {
    int val;
    struct semid_ds *buf;
    unsigned short  *array;
};

struct sembuf v = { 0, 1, 0};

void do_work(int id, union semun u){
    int temp=semctl(id, 0, GETVAL, u);
    if(temp>=5){
        return;
    }
    while(1){
        if (semctl(id, 0, GETVAL, u) == temp) {
            if (semop(id, &v, 1) < 0) {
                perror("semop p");
                exit(13);
            }
        }
        if (semctl(id, 0, GETVAL, u) > temp) {
            printf("VAL = %d\n", semctl(id, 0, GETVAL, u));
            break;
        }
    }
}

int main()
{
    int id = semget(IPC_PRIVATE, 1, 0600);
    if(id < 0)
    {
        perror("semget"); exit(11);
    }
    union semun u;
    u.val = 0;
    printf("VAL = %d\n", semctl(id, 0, GETVAL, u));
    int a, b, c;
    a=fork();
    b=fork();
    c=fork();

    sleep(1);
    if(a>0) {
        do_work(id,u);
    }
    sleep(3);
    if(b>0) {
        do_work(id,u);
    }
```

```
    sleep(2);
    if(c>0) {
        do_work(id,u);
    }
}
```

4.5 Source code for Exercise Five

4.5.1 Part 1

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <unistd.h>

union semun {
    int val;
    struct semid_ds *buf;
    unsigned short  *array;
};

struct sembuf p = { 0, -1, SEM_UNDO};
struct sembuf v = { 0, +1, SEM_UNDO};

int main()
{
    int key=ftok("/temp",8);
    int id = semget(key, 1, 0666 | IPC_CREAT);
    if(id < 0)
    {
        perror("semget"); exit(11);
    }
    union semun u;
    u.val = 1;

    int fd2;

    char * myfifo = "/tmp/myfifo";

    mkfifo(myfifo, 0666);

    int n;
    while (1)
    {
        // First open in read only and read
        fd2 = open(myfifo,O_RDONLY);

        if(semop(id, &p, 1) < 0)
        {
            perror("semop p"); exit(13);
        } else{
```

```
            read(fd2, &n, sizeof(n));
            printf("Op success\n");
        }



        // Print the read string and close
        printf("User1: %d\n", n);
        if(n>20){
            break;
        }
        close(fd2);
        sleep(1);
    }
    if(unlink("myfifo")<0)
        perror("unlink");
    return 0;
}
```

4.5.2 Part 2

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/sem.h>

union semun {
    int val;
    struct semid_ds *buf;
    unsigned short  *array;
};

struct sembuf p = { 0, -1, SEM_UNDO};
struct sembuf v = { 0, +1, SEM_UNDO};

int main()
{
    int key=ftok("/temp",8);
    int id = semget(key, 1, 0666 | IPC_CREAT);
    if(id < 0)
    {
        perror("semget"); exit(11);
    }
    union semun u;
    u.val = 1;

    int fd1;
    int i=1;

    char * myfifo = "/tmp/myfifo";

    mkfifo(myfifo, 0666);
```

```
    char arr1[80], arr2[80];
    while (i<22)
    {
        i+=2;
        fd1 = open(myfifo, O_WRONLY);
        if(semop(id, &v, 1) < 0)
        {
            perror("semop p"); exit(14);
        } else{
            write(fd1, &i, sizeof(i));
        }

        sleep(1);
        printf("P1 wrote %d\n",i);
        close(fd1);
    }
    return 0;
}
```

4.6 Source code for Exercise Six

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/ipc.h>
#include <sys/sem.h>

const int N_PHILOSOPHERS  =  5;
const int MEALS_TO_HEAVEN = 10;
const int MAX_DELAY = 500000;

int chopsticks;
int not_at_table;

int philosopher(int n);

int main(){
    int i, status;
    pid_t phil[N_PHILOSOPHERS];
    printf("The Dining-Philosophers Problem\n");

    chopsticks = semget(IPC_PRIVATE, N_PHILOSOPHERS, IPC_CREAT | 0600);
    for(i=0; i<N_PHILOSOPHERS; i++){
        semctl(chopsticks, i, SETVAL, 1);
    }
    not_at_table = semget(IPC_PRIVATE, 1, IPC_CREAT | 0600);

    semctl(not_at_table, 0, SETVAL, 5);

    for(i=0; i < N_PHILOSOPHERS; i++){
        int pid = fork();
```

```
        if(pid == 0){
            int ret = philosopher(i);
            exit(ret);
        }
        else{
            phil[i] = pid;
        }
    }


    for(i = 0; i < N_PHILOSOPHERS; i++) {
        waitpid(phil[i], &status, 0);
    }

    semctl(chopsticks, 0, IPC_RMID, 0);
    semctl(not_at_table, 0, IPC_RMID, 0);
    return 0;
}


int philosopher(int n){
    int i, j, first, second;
    struct sembuf op;
    op.sem_flg = 0;
    srand(n);

    //Create deadlock by using same order of chopstick request for
first and last philosophers
    first =  (n < N_PHILOSOPHERS)? n     : 0;
    second = (n < N_PHILOSOPHERS)? n     : N_PHILOSOPHERS-1; //Changing
'n' to 'n+1' will solve the issue of the deadlock


    op.sem_op = -1;
    op.sem_num = 0;
    semop(not_at_table, &op, 1);
    printf("Philosopher %d at the table\n", n);

    op.sem_op = 0;
    op.sem_num = 0;
    semop(not_at_table, &op, 1);

    for(i = 0; i < MEALS_TO_HEAVEN; i++) {
        int sleep_time = rand() % MAX_DELAY;
        usleep(sleep_time);

        printf("%2d: Philosopher %d is thinking ...\n", i,n);

        op.sem_op = -1;
        op.sem_num = first;
        semop(chopsticks, &op, 1);

        op.sem_op = -1;
        op.sem_num = second;
        semop(chopsticks, &op, 1);
```

```
        printf("%2d: Philosopher %d is eating ...\n", i,n);

        op.sem_op = +1;
        op.sem_num = first;
        semop(chopsticks, &op, 1);

        op.sem_op = +1;
        op.sem_num = second;
        semop(chopsticks, &op, 1);
    }

    printf("Philosopher %d going to heaven\n",n);
    exit(n);
}
```

4.7 SOURCE CODE FOR EXERCISE SEVEN

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


union semun {
    int val;
    struct semid_ds *buf;
    unsigned short  *array;
};

struct sembuf p = { 0, -1, SEM_UNDO};
struct sembuf v = { 0, +1, SEM_UNDO};

int main()
{
    int id = semget(IPC_PRIVATE, 1, 0666 | IPC_CREAT);
    if(id < 0)
    {
        perror("semget"); exit(11);
    }
    union semun u;
    u.val = 1;
    if(semctl(id, 0, SETVAL, u) < 0)
    {
        perror("semctl"); exit(12);
    }
    int pid;
    pid =  fork();
    if(pid < 0)
    {
        perror("fork"); exit(1);
    }
```

```
    else if(pid)
    {
        printf("Val 2.1 = %d\n",semctl(id, 0, GETVAL, u));
        for(int i = 0; i < 5; ++i)
        {
            if(semop(id, &p, 1) < 0)
            {
                perror("semop p"); exit(13);
            }
            printf("Child Process\n");
            if(semop(id, &v, 1) < 0)
            {
                perror("semop p"); exit(14);
            }
        }
        if(semop(id, &p, 1) < 0)
        {
            perror("semop p"); exit(13);
        }
        printf("Val 2.2 = %d\n",semctl(id, 0, GETVAL, u));
    }
    else
    {
        printf("Val 1.1 = %d\n",semctl(id, 0, GETVAL, u));
        for(int i = 0; i < 5; ++i)
        {
            if(semop(id, &p, 1) < 0)
            {
                perror("semop p"); exit(15);
            }
            printf("Parent Process\n");
            if(semop(id, &v, 1) < 0)
            {
                perror("semop p"); exit(16);
            }
        }
        printf("Val 1.2 = %d\n",semctl(id, 0, GETVAL, u));
    }
    sleep(1);
}
```

4.8 SOURCE CODE FOR EXERCISE EIGHT

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


union semun {
    int val;
    struct semid_ds *buf;
```

```
    unsigned short *array;
}arg;

struct sembuf p = { 0, -1, SEM_UNDO};
struct sembuf v = { 0, +1, SEM_UNDO};

int main()
{
    int id = semget(IPC_PRIVATE, 1, 0666 | IPC_CREAT);
    if(id < 0)
    {
        perror("semget"); exit(11);
    }
    union semun u;
    u.val = 1;
    short unsigned int a=2;
    u.array = &a;
    if(semctl(id, 0, SETVAL, u) < 0)
    {
        perror("semctl"); exit(12);
    }
    int pid;
    pid =  fork();
    if(pid < 0)
    {
        perror("fork"); exit(1);
    }
    else if(pid)
    {
        for(int i = 0; i < 5; ++i)
        {
            if(semop(id, &p, 1) < 0)
            {
                perror("semop p"); exit(13);
            }
            printf("Child Process\n");
            if(semop(id, &v, 1) < 0)
            {
                perror("semop p"); exit(14);
            }
        }
    }
    else
    {
        for(int i = 0; i < 5; ++i)
        {
            if(semop(id, &p, 1) < 0)
            {
                perror("semop p"); exit(15);
            }
            printf("Parent Process\n");
            if(semop(id, &v, 1) < 0)
            {
                perror("semop p"); exit(16);
            }
```

```
        }

        int x= semctl(id, 0, SETALL, u);
        printf("SETALL = %d\n", x);
        int y= semctl(id, 0, GETALL, u);
        printf("GETALL = %d\n", y);

    }
    sleep(1);
}
```

4.9 SOURCE CODE FOR EXERCISE NINE

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

#define TEXT_LENGTH 50

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

union semun {
    int val;
    struct semid_ds *buf;
    unsigned short  *array;
};

struct sembuf p = { 0, -1, SEM_UNDO};
struct sembuf v = { 0, +1, SEM_UNDO};

void send_message(int mqid, char message[], int id_1)
{
    if(semop(id_1, &p, 1) < 0)
    {
        perror("semop p"); exit(13);
    }else{
        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, int id_1)
{
    if(semop(id_1, &v, 1) < 0)
    {
        perror("semop p"); exit(13);
    } else{
        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_1 = semget(IPC_PRIVATE, 1, 0666 | IPC_CREAT);
    if(id_1 < 0)
    {
        perror("semget"); exit(11);
    }
    union semun u;
    u.val = 1;

    int id = msgget(IPC_PRIVATE, 0600 | IPC_NOWAIT);
    int a=fork();

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


    if(a!=0){
        receive_message(id, id_1);
    }
    else{
        send_message(id,"Operating Systems",id_1);
    }

    remove_queue(id);

    return 0;
}
```

4.10 Source code for Exercise Ten

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/wait.h>

union semun {
    int val;
    struct semid_ds *buf;
    unsigned short  *array;
};

struct sembuf p = { 0, -1, 0};
struct sembuf v = { 0, +1, 0};

void child_process(int fd[]){
    close(fd[0]);
    int x;
    for(x=1;x<=10;x++)
        write(fd[1], &x, sizeof(int));
    close(fd[1]);
}

int parent_process(int fd[]){
    close(fd[1]);
    int y;
    for(int x=1;x<=10;x++) {
        read(fd[0], &y, sizeof(int));
        printf("Got number from child process: %d\n", y);
    }
    close(fd[0]);
    return y;
}

int main(int argc, char *argv[]){
    int id_1 = semget(IPC_PRIVATE, 1, 0666 | IPC_CREAT);
    if(id_1 < 0)
    {
        perror("semget"); exit(11);
    }
    union semun u;
    u.val = 1;

    int fd[2];

    if(pipe(fd)==-1){
        printf("Error occurred\n");
        return 1;
```

```
    }
    int id=fork();
    if(id<0)
        perror("fork error");


    if(id==0){
        child_process(fd);
        if(semop(id_1, &v, 1) < 0)
        {
            perror("semop p"); exit(13);
        } else{
            return 0;
        }
    }
    else{
        int x = parent_process(fd);
        if(x==10){
            if(semop(id_1, &p, 1) < 0)
            {
                perror("semop p"); exit(13);
            } else{
                printf("END\n");
            }
        }
    }
    return 0;
}
```