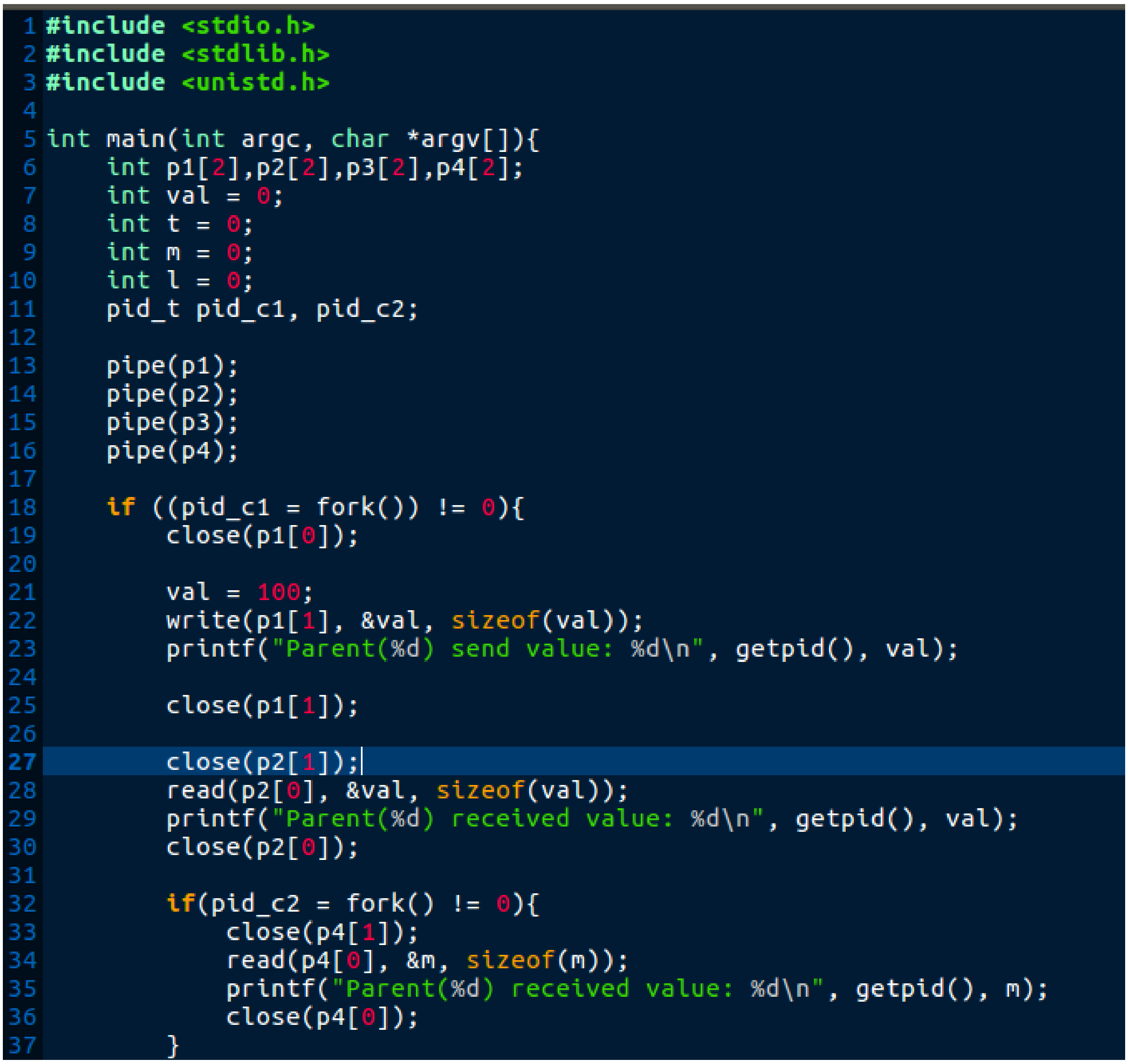
Advanced Unix Programming
Assignment-10

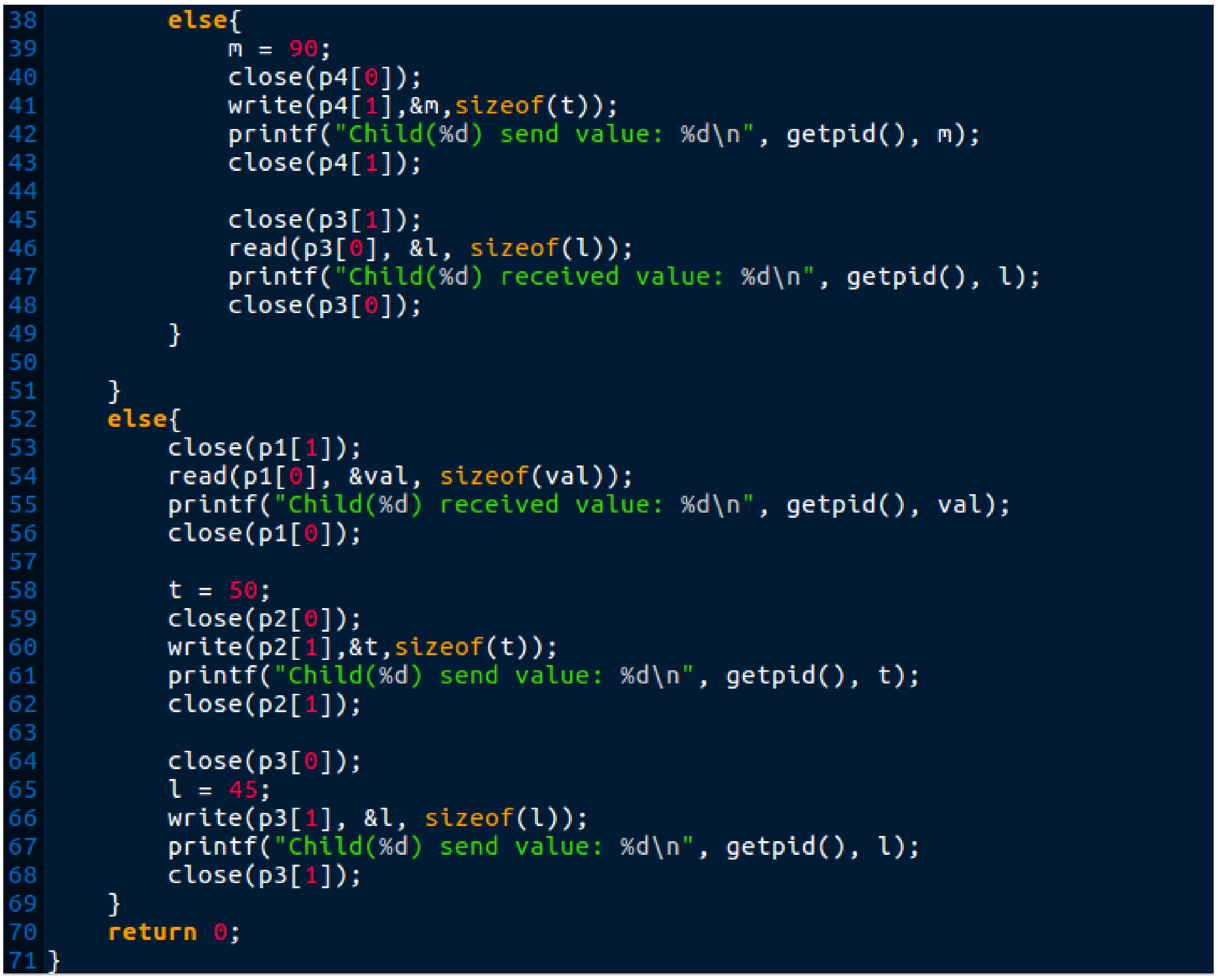
	Nikhil Gawande	**111408013**
	Anupam Godse	**111408016**

Q1. A pipe setup is given below that involves three processes. P is the parent process, and C1 and C2 are child processes, spawned from P. The pipes are named p1, p2, p3, and p4. Write a program that establishes the necessary pipe connections, setups, and carries out the reading/writing of the text in the indicated directions.

CODE:

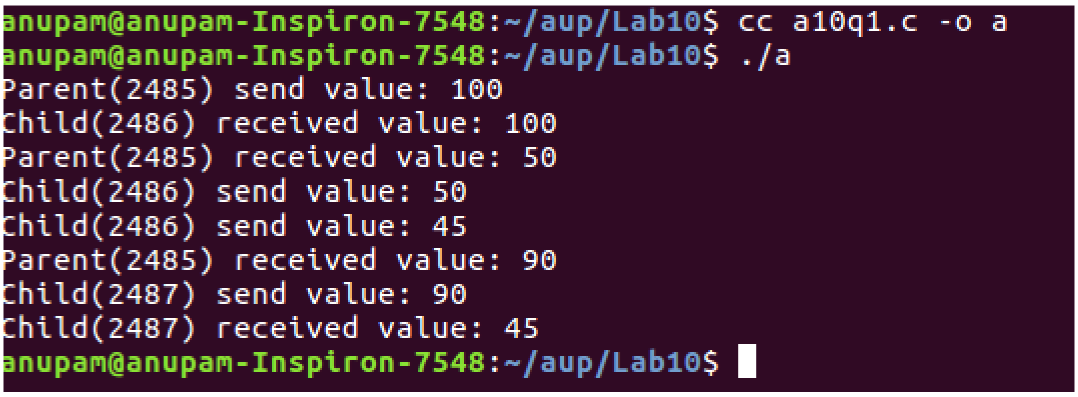


```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[]){
    int p1[2],p2[2],p3[2],p4[2];
    int val = 0;
    int t = 0;
    int m = 0;
    int l = 0;
    pid_t pid_c1, pid_c2;

    pipe(p1);
    pipe(p2);
    pipe(p3);
    pipe(p4);

    if ((pid_c1 = fork()) != 0){
        close(p1[0]);

        val = 100;
        write(p1[1], &val, sizeof(val));
        printf("Parent(%d) send value: %d\n", getpid(), val);

        close(p1[1]);

        close(p2[1]);
        read(p2[0], &val, sizeof(val));
        printf("Parent(%d) received value: %d\n", getpid(), val);
        close(p2[0]);

        if(pid_c2 = fork() != 0){
            close(p4[1]);
            read(p4[0], &m, sizeof(m));
            printf("Parent(%d) received value: %d\n", getpid(), m);
            close(p4[0]);
        }
```



```c
        else{
            m = 90;
            close(p4[0]);
            write(p4[1],&m,sizeof(t));
            printf("Child(%d) send value: %d\n", getpid(), m);
            close(p4[1]);

            close(p3[1]);
            read(p3[0], &l, sizeof(l));
            printf("Child(%d) received value: %d\n", getpid(), l);
            close(p3[0]);
        }

    }
    else{
        close(p1[1]);
        read(p1[0], &val, sizeof(val));
        printf("Child(%d) received value: %d\n", getpid(), val);
        close(p1[0]);

        t = 50;
        close(p2[0]);
        write(p2[1],&t,sizeof(t));
        printf("Child(%d) send value: %d\n", getpid(), t);
        close(p2[1]);

        close(p3[0]);
        l = 45;
        write(p3[1], &l, sizeof(l));
        printf("Child(%d) send value: %d\n", getpid(), l);
        close(p3[1]);
    }
    return 0;
}
```

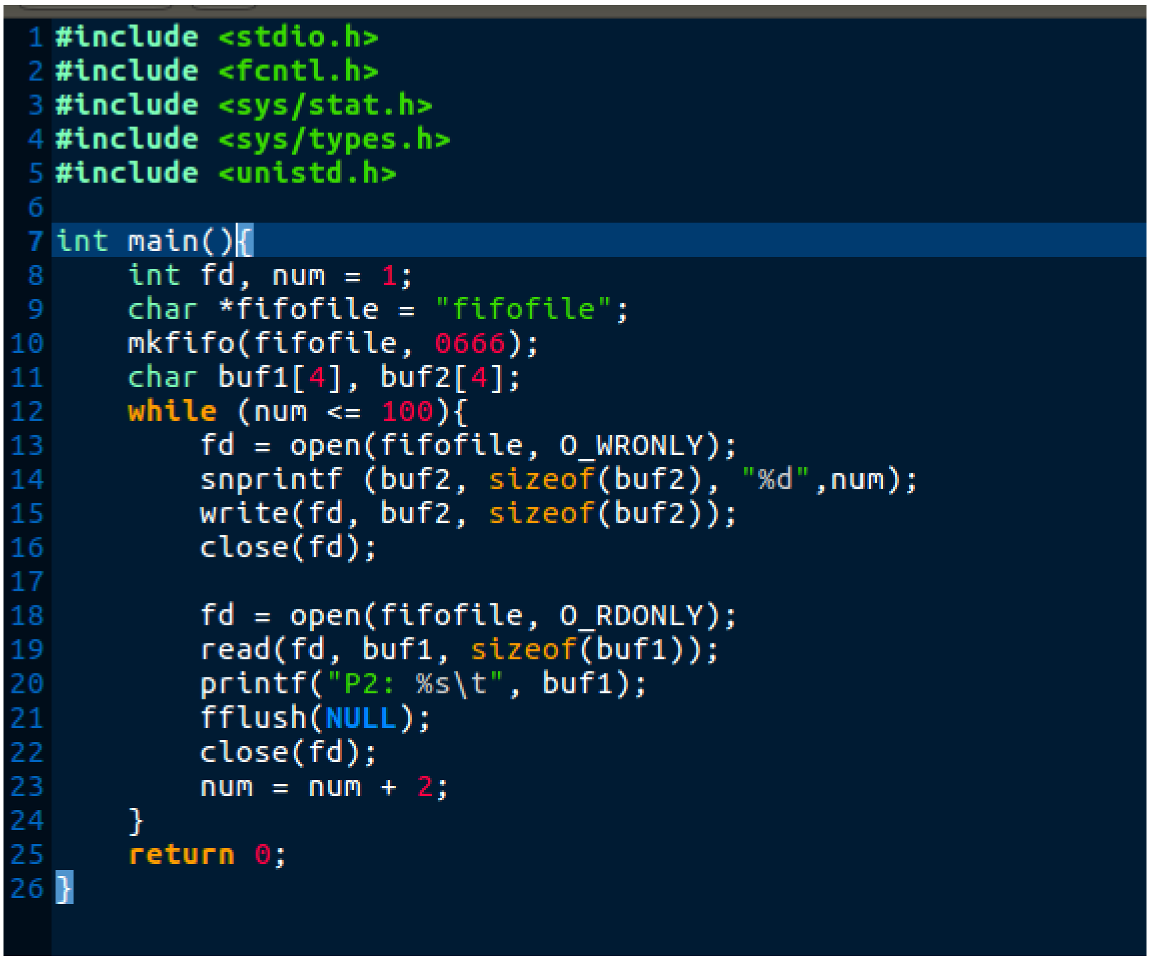
OUTPUT:



```
anupam@anupam-Inspiron-7548:~/aup/Lab10$ cc a10q1.c -o a
anupam@anupam-Inspiron-7548:~/aup/Lab10$ ./a
Parent(2485) send value: 100
Child(2486) received value: 100
Parent(2485) received value: 50
Child(2486) send value: 50
Child(2486) send value: 45
Parent(2485) received value: 90
Child(2487) send value: 90
Child(2487) received value: 45
anupam@anupam-Inspiron-7548:~/aup/Lab10$ 
```

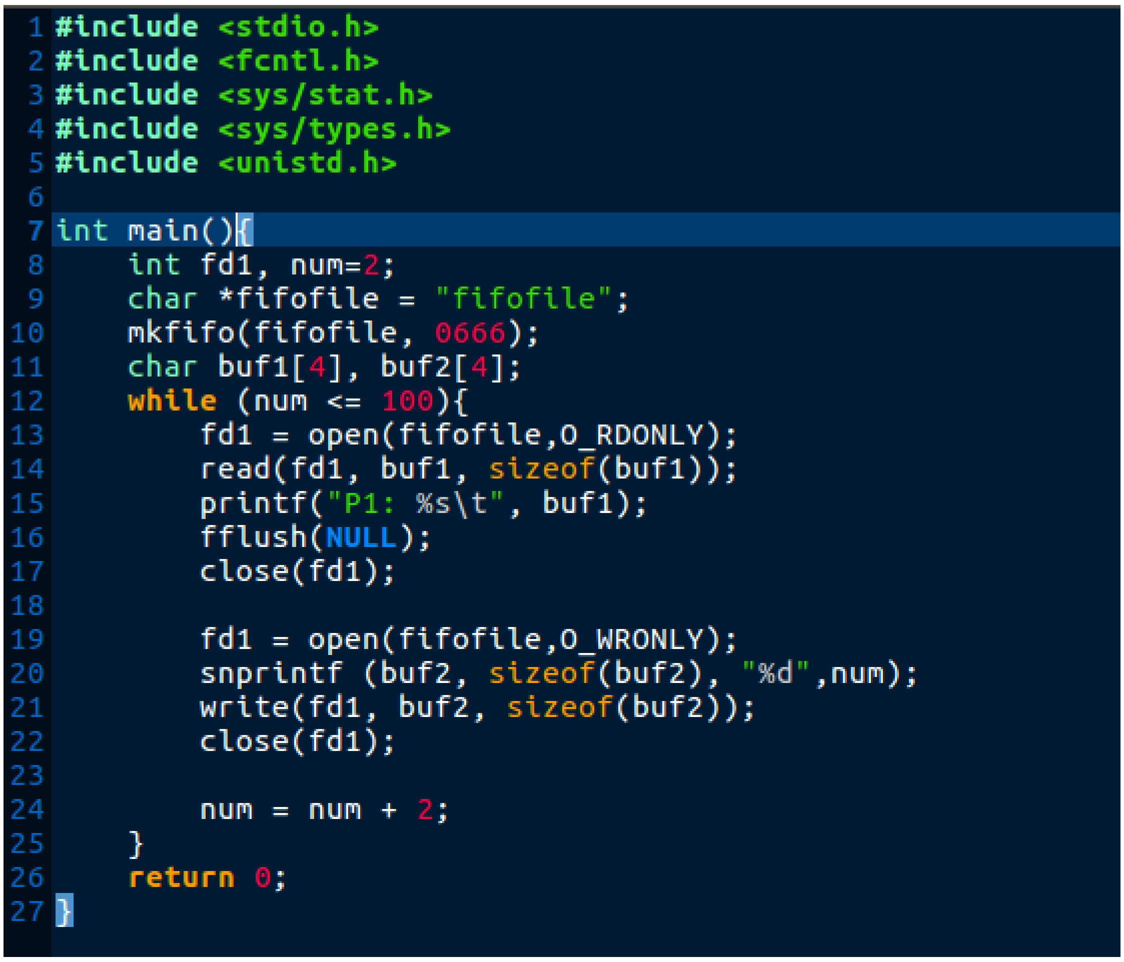
Q2. Let P1 and P2 be two processes alternatively writing numbers from 1 to 100 to a file. Let P1 write odd numbers and p2, even. Implement the synchronization between the processes using FIFO.

CODE: Process 1:



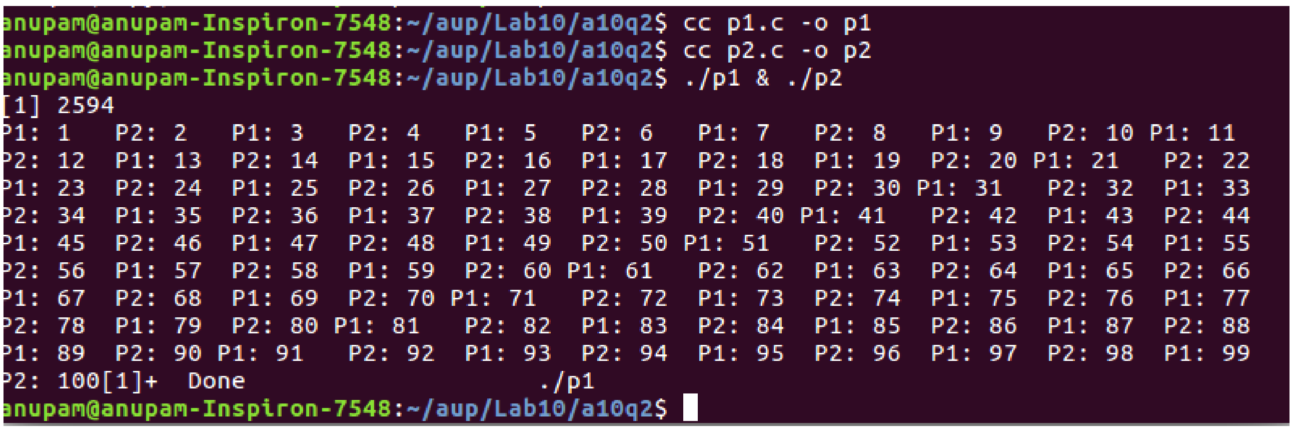
```c
#include <stdio.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int main(){
    int fd, num = 1;
    char *fifofile = "fifofile";
    mkfifo(fifofile, 0666);
    char buf1[4], buf2[4];
    while (num <= 100){
        fd = open(fifofile, O_WRONLY);
        snprintf (buf2, sizeof(buf2), "%d",num);
        write(fd, buf2, sizeof(buf2));
        close(fd);

        fd = open(fifofile, O_RDONLY);
        read(fd, buf1, sizeof(buf1));
        printf("P2: %s\t", buf1);
        fflush(NULL);
        close(fd);
        num = num + 2;
    }
    return 0;
}
```

Process P2:



```c
#include <stdio.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int main(){
    int fd1, num=2;
    char *fifofile = "fifofile";
    mkfifo(fifofile, 0666);
    char buf1[4], buf2[4];
    while (num <= 100){
        fd1 = open(fifofile,O_RDONLY);
        read(fd1, buf1, sizeof(buf1));
        printf("P1: %s\t", buf1);
        fflush(NULL);
        close(fd1);

        fd1 = open(fifofile,O_WRONLY);
        snprintf (buf2, sizeof(buf2), "%d",num);
        write(fd1, buf2, sizeof(buf2));
        close(fd1);

        num = num + 2;
    }
    return 0;
}
```

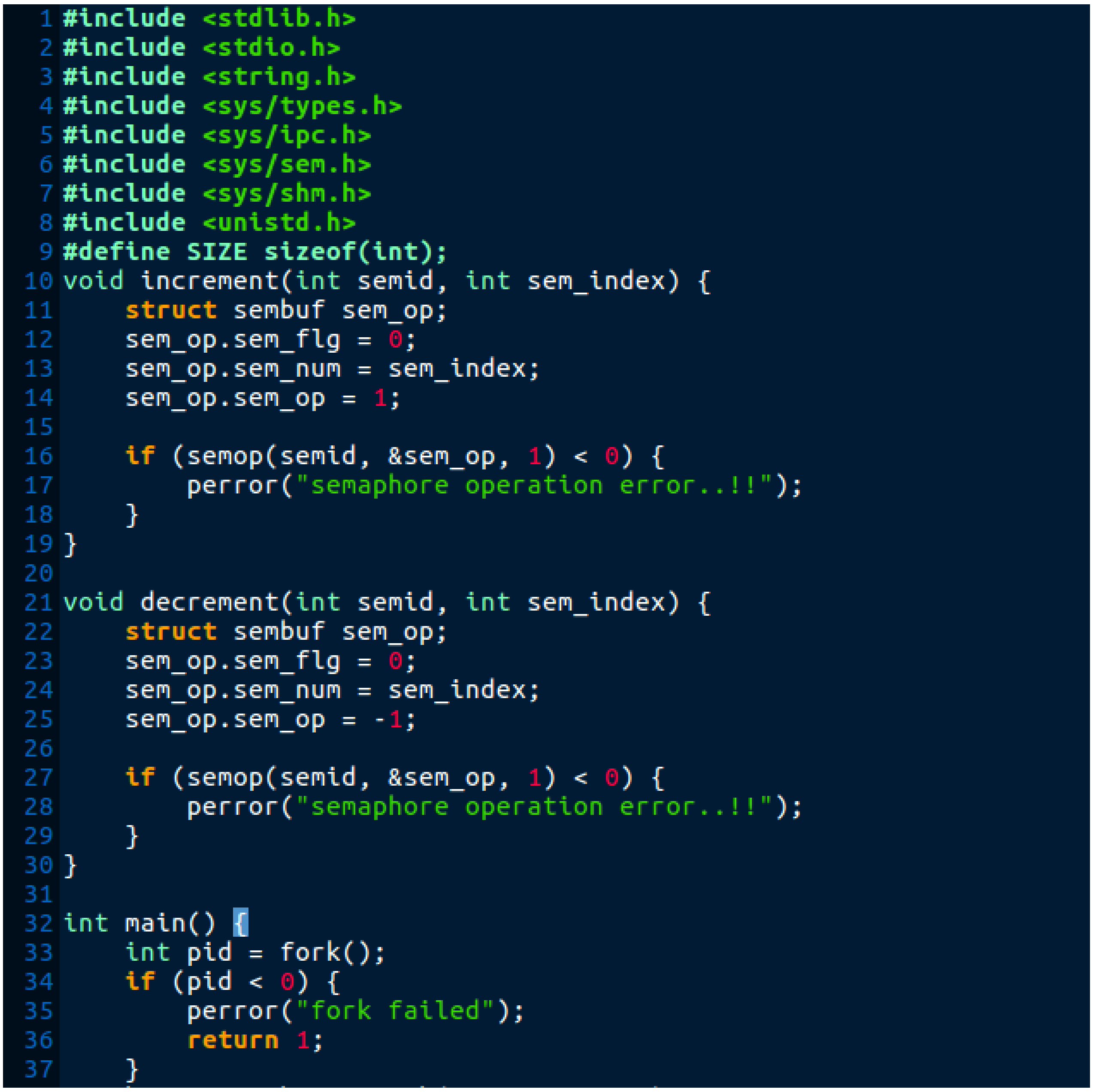
OUTPUT:



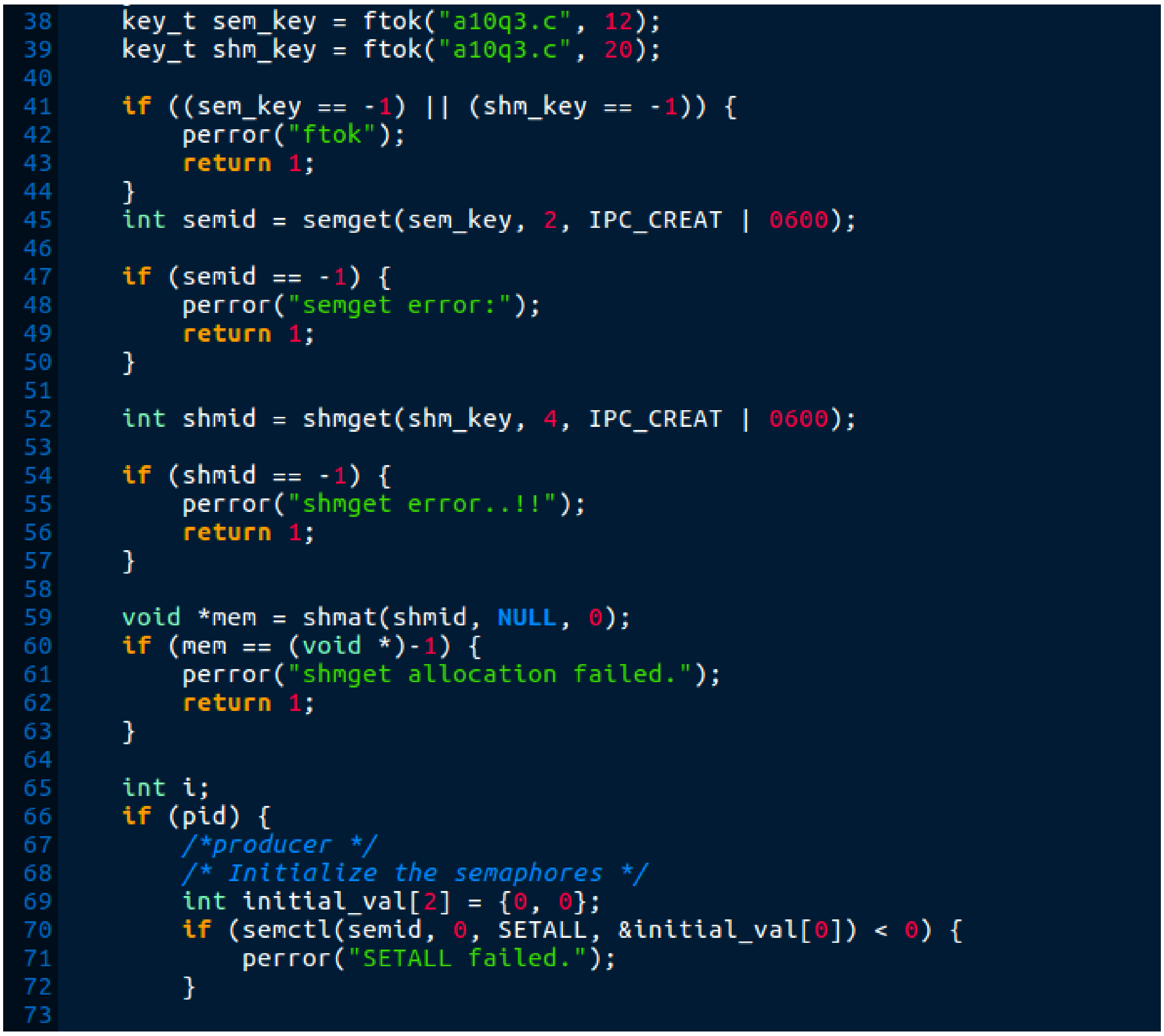
```
anupam@anupam-Inspiron-7548:~/aup/Lab10/a10q2$ cc p1.c -o p1
anupam@anupam-Inspiron-7548:~/aup/Lab10/a10q2$ cc p2.c -o p2
anupam@anupam-Inspiron-7548:~/aup/Lab10/a10q2$ ./p1 & ./p2
[1] 2594
P1: 1    P2: 2    P1: 3    P2: 4    P1: 5    P2: 6    P1: 7    P2: 8    P1: 9    P2: 10 P1: 11
P2: 12   P1: 13   P2: 14   P1: 15   P2: 16   P1: 17   P2: 18   P1: 19   P2: 20 P1: 21    P2: 22
P1: 23   P2: 24   P1: 25   P2: 26   P1: 27   P2: 28   P1: 29   P2: 30 P1: 31    P2: 32   P1: 33
P2: 34   P1: 35   P2: 36   P1: 37   P2: 38   P1: 39   P2: 40 P1: 41    P2: 42   P1: 43   P2: 44
P1: 45   P2: 46   P1: 47   P2: 48   P1: 49   P2: 50 P1: 51    P2: 52   P1: 53   P2: 54   P1: 55
P2: 56   P1: 57   P2: 58   P1: 59   P2: 60 P1: 61    P2: 62   P1: 63   P2: 64   P1: 65   P2: 66
P1: 67   P2: 68   P1: 69   P2: 70 P1: 71    P2: 72   P1: 73   P2: 74   P1: 75   P2: 76   P1: 77
P2: 78   P1: 79   P2: 80 P1: 81    P2: 82   P1: 83   P2: 84   P1: 85   P2: 86   P1: 87   P2: 88
P1: 89   P2: 90 P1: 91    P2: 92   P1: 93   P2: 94   P1: 95   P2: 96   P1: 97   P2: 98   P1: 99
P2: 100[1]+  Done                    ./p1
anupam@anupam-Inspiron-7548:~/aup/Lab10/a10q2$ 
```

Q3.Implement a producer-consumer setup using shared memory and semaphore. Ensure that data doesn’t get over-written by the producer before the consumer reads and displays on the screen. Also ensure that the consumer doesn’t read the same data twice.

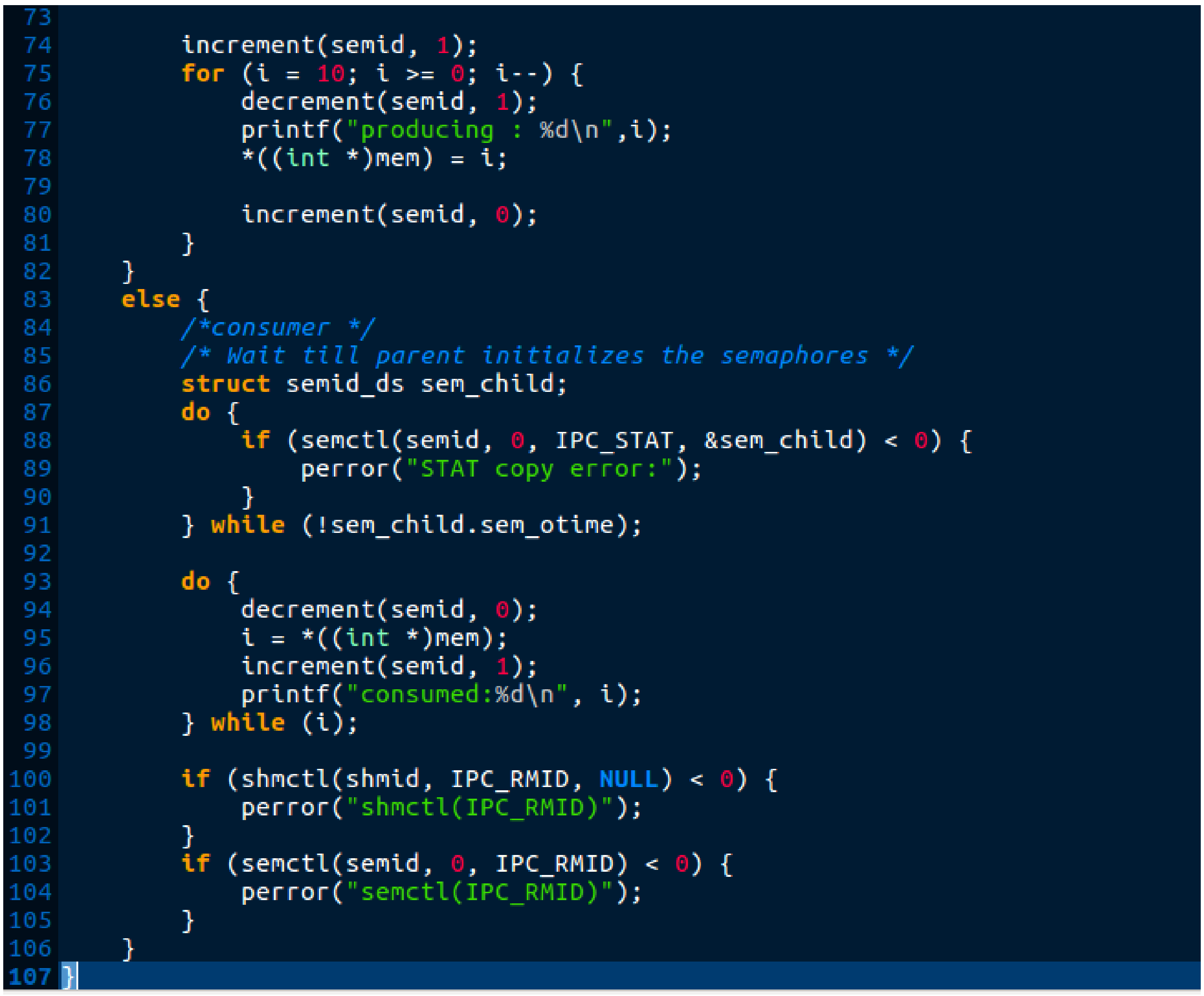
CODE:



```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/shm.h>
#include <unistd.h>
#define SIZE sizeof(int);
void increment(int semid, int sem_index) {
    struct sembuf sem_op;
    sem_op.sem_flg = 0;
    sem_op.sem_num = sem_index;
    sem_op.sem_op = 1;

    if (semop(semid, &sem_op, 1) < 0) {
        perror("semaphore operation error..!!");
    }
}

void decrement(int semid, int sem_index) {
    struct sembuf sem_op;
    sem_op.sem_flg = 0;
    sem_op.sem_num = sem_index;
    sem_op.sem_op = -1;

    if (semop(semid, &sem_op, 1) < 0) {
        perror("semaphore operation error..!!");
    }
}

int main() {
    int pid = fork();
    if (pid < 0) {
        perror("fork failed");
        return 1;
    }
```

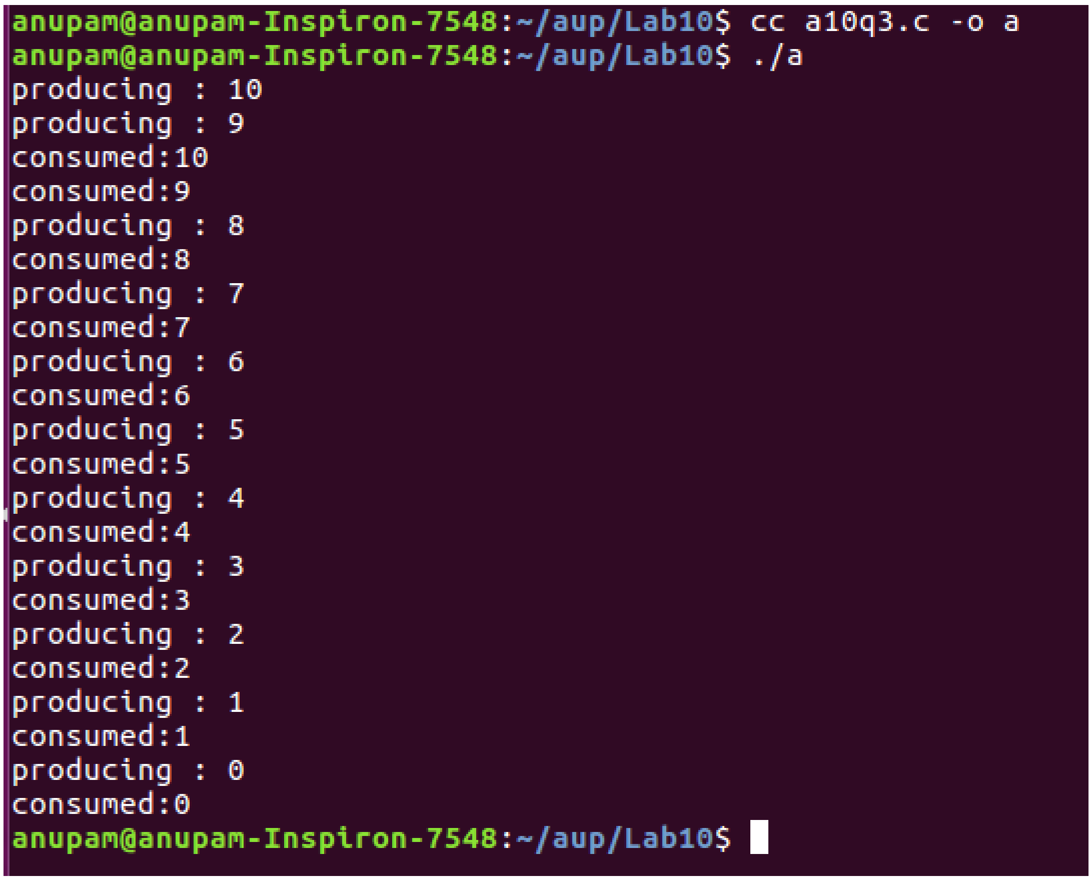


```c
    key_t sem_key = ftok("a10q3.c", 12);
    key_t shm_key = ftok("a10q3.c", 20);

    if ((sem_key == -1) || (shm_key == -1)) {
        perror("ftok");
        return 1;
    }
    int semid = semget(sem_key, 2, IPC_CREAT | 0600);

    if (semid == -1) {
        perror("semget error:");
        return 1;
    }

    int shmid = shmget(shm_key, 4, IPC_CREAT | 0600);

    if (shmid == -1) {
        perror("shmget error..!!");
        return 1;
    }

    void *mem = shmat(shmid, NULL, 0);
    if (mem == (void *)-1) {
        perror("shmget allocation failed.");
        return 1;
    }

    int i;
    if (pid) {
        /*producer */
        /* Initialize the semaphores */
        int initial_val[2] = {0, 0};
        if (semctl(semid, 0, SETALL, &initial_val[0]) < 0) {
            perror("SETALL failed.");
        }

```



```c

        increment(semid, 1);
        for (i = 10; i >= 0; i--) {
            decrement(semid, 1);
            printf("producing : %d\n",i);
            *((int *)mem) = i;

            increment(semid, 0);
        }
    }
    else {
        /*consumer */
        /* Wait till parent initializes the semaphores */
        struct semid_ds sem_child;
        do {
            if (semctl(semid, 0, IPC_STAT, &sem_child) < 0) {
                perror("STAT copy error:");
            }
        } while (!sem_child.sem_otime);

        do {
            decrement(semid, 0);
            i = *((int *)mem);
            increment(semid, 1);
            printf("consumed:%d\n", i);
        } while (i);

        if (shmctl(shmid, IPC_RMID, NULL) < 0) {
            perror("shmctl(IPC_RMID)");
        }
        if (semctl(semid, 0, IPC_RMID) < 0) {
            perror("semctl(IPC_RMID)");
        }
    }
}
```

Output:



```
anupam@anupam-Inspiron-7548:~/aup/Lab10$ cc a10q3.c -o a
anupam@anupam-Inspiron-7548:~/aup/Lab10$ ./a
producing : 10
producing : 9
consumed:10
consumed:9
producing : 8
consumed:8
producing : 7
consumed:7
producing : 6
consumed:6
producing : 5
consumed:5
producing : 4
consumed:4
producing : 3
consumed:3
producing : 2
consumed:2
producing : 1
consumed:1
producing : 0
consumed:0
anupam@anupam-Inspiron-7548:~/aup/Lab10$ 
```