Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчет по лабораторной работе №5 по курсу «Операционные системы»

Тема Взаимодействие параллельных процессов

Студент Кононенко С.С.

Группа ИУ7-53Б

Оценка (баллы) ____________________________

Преподаватели Рязанова Н.Ю.

Москва — 2020 г.

Задача «Производство – потребление»

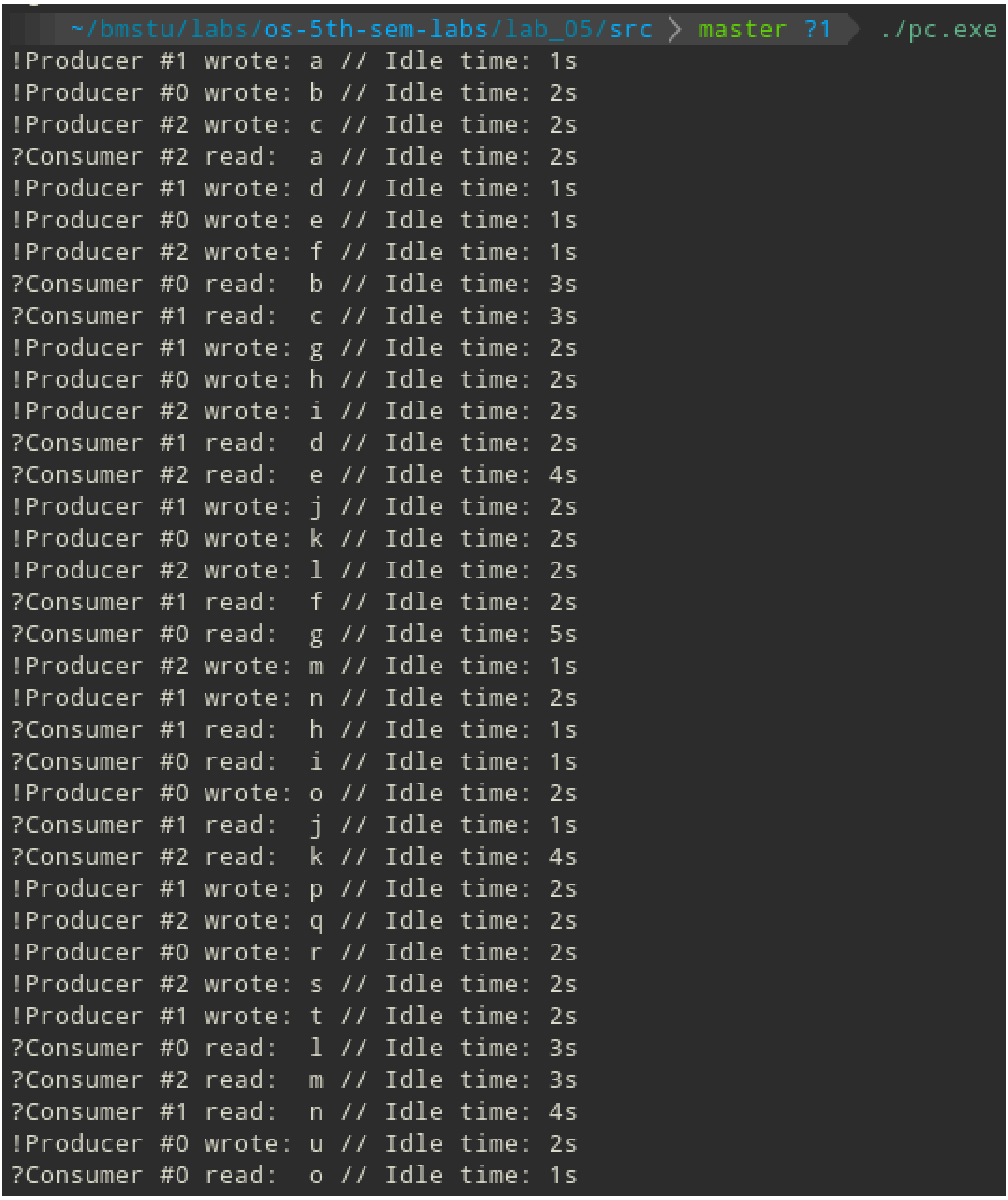
Листинг 1 – Реализация задачи «производство – потребление»

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <time.h>
#include <wait.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <sys/types.h>

#define N 24
#define MAX_SEMS 3
#define ITERS 8

#define P_COUNT 3
#define C_COUNT 3

#define BIN_SEM 0
#define BUF_FULL 1
#define BUF_EMPTY 2

#define MAX_RAND_P 2
#define MAX_RAND_C 5

typedef char data_t[N];
typedef struct
{
    size_t r_pos;
    size_t w_pos;
    data_t data;
} cbuf_t;

struct sembuf P_LOCK[2] = {{BUF_EMPTY, -1, 0}, {BIN_SEM, -1, 0}};
struct sembuf P_RELEASE[2] = {{BUF_FULL, 1, 0}, {BIN_SEM, 1, 0}};

struct sembuf C_LOCK[2] = {{BUF_FULL, -1, 0}, {BIN_SEM, -1, 0}};
struct sembuf C_RELEASE[2] = {{BUF_EMPTY, 1, 0}, {BIN_SEM, 1, 0}};

int init_buf(cbuf_t *const buf)
```

```c
{
    if (!buf)
    {
        return -1;
    }
    memset(buf, 0, sizeof(cbuf_t));

    return 0;
}

int write_buf(cbuf_t *const buf, const char c)
{
    if (!buf)
    {
        return -1;
    }
    buf->data[buf->w_pos++] = c;
    buf->w_pos %= N;

    return 0;
}

int read_buf(cbuf_t *const buf, char *const dst)
{
    if (!buf)
    {
        return -1;
    }
    *dst = buf->data[buf->r_pos++];
    buf->r_pos %= N;

    return 0;
}

int p_run(cbuf_t *const buf, const int s_id, const int p_id)
{
    if (!buf)
    {
        return -1;
    }

    srand(time(NULL) + p_id);

    int stime;
    char ch;

    for (size_t i = 0; i < ITERS; ++i)
    {
        stime = rand() % MAX_RAND_P + 1;
        sleep(stime);

        if (semop(s_id, P_LOCK, 2) == -1)
        {
            perror("Producer␣lock␣error.");

            exit(EXIT_FAILURE);
        }

        ch = 'a' + (char)(buf->w_pos % 26);

        if (write_buf(buf, ch) == -1)
        {
            perror("Buffer␣write␣error.");

            return EXIT_FAILURE;
        }
        printf("!Producer␣#%d␣wrote:␣%c␣//␣Idle␣time:␣%ds\n", p_id, ch, stime);

        if (semop(s_id, P_RELEASE, 2) == -1)
        {
            perror("Producer␣release␣error.");

            exit(EXIT_FAILURE);
        }
    }

    return EXIT_SUCCESS;
}

int c_run(cbuf_t *const buf, const int s_id, const int c_id)
{
    if (!buf)
    {
        return -1;
    }

    srand(time(NULL) + c_id + P_COUNT);

    int stime;
    char ch;

    for (size_t i = 0; i < ITERS; ++i)
    {
        stime = rand() % MAX_RAND_C + 1;
        sleep(stime);
```

```c

        if (semop(s_id, C_LOCK, 2) == -1)
        {
            perror("Consumer␣lock␣error.");

            exit(EXIT_FAILURE);
        }

        if (read_buf(buf, &ch) == -1)
        {
            perror("Buffer␣read␣error.");

            return EXIT_FAILURE;
        }
        printf("?Consumer␣#%d␣read:␣␣%c␣//␣Idle␣time:␣%ds\n", c_id, ch, stime);

        if (semop(s_id, C_RELEASE, 2) == -1)
        {
            perror("Consumer␣release␣error.");

            exit(EXIT_FAILURE);
        }
    }

    return EXIT_SUCCESS;
}

int main()
{
    setbuf(stdout, NULL);

    int fd = shmget(IPC_PRIVATE, sizeof(cbuf_t), IPC_CREAT | S_IRWXU | S_IRWXG | S_IRWXO);
    if (fd == -1)
    {
        perror("shmget␣failed.");

        return EXIT_FAILURE;
    }

    cbuf_t *buf = shmat(fd, 0, 0);
    if (buf == (void *)-1)
    {
        perror("shmat␣failed.");

        return EXIT_FAILURE;
    }

    if (init_buf(buf) == -1)
    {
        perror("Buffer␣initialization␣failed.");

        return EXIT_FAILURE;
    }

    int s_id = semget(IPC_PRIVATE, MAX_SEMS, IPC_CREAT | S_IRWXU | S_IRWXG
        | S_IRWXO);
    if (s_id == -1)
    {
        perror("semget␣failed.");

        return EXIT_FAILURE;
    }

    semctl(s_id, BIN_SEM, SETVAL, 1);
    semctl(s_id, BUF_EMPTY, SETVAL, N);
    semctl(s_id, BUF_FULL, SETVAL, 0);

    int chpid;
    for (size_t i = 0; i < P_COUNT; ++i)
    {
        switch ((chpid = fork()))
        {
        case -1:
            perror("Producer␣fork␣failed.");

            exit(EXIT_FAILURE);
            break;
        case 0:
            p_run(buf, s_id, i);

            return EXIT_SUCCESS;
        }
    }

    for (size_t i = 0; i < C_COUNT; ++i)
    {
        switch ((chpid = fork()))
        {
        case -1:
            perror("Consumer␣fork␣failed.");

            exit(EXIT_FAILURE);
            break;
        case 0:
```

```c
            c_run(buf, s_id, i);
            return EXIT_SUCCESS;
        }
    }

    for (size_t i = 0; i < C_COUNT + P_COUNT; ++i)
    {
        int status;
        if (wait(&status) == -1)
        {
            perror("Child␣error.");

            exit(EXIT_FAILURE);
        }
        if (!WIFEXITED(status))
        {
            printf("Child␣process␣terminated␣abnormally\n");
        }
    }

    if (shmdt((void *)buf) == -1 ||
        shmctl(fd, IPC_RMID, NULL) == -1 ||
        semctl(s_id, IPC_RMID, 0) == -1)
    {
        perror("Exit␣error.");

        return EXIT_FAILURE;
    }

    return EXIT_SUCCESS;
}
```



```
 ~/bmstu/labs/os-5th-sem-labs/lab_05/src 〉 master ?1   ./pc.exe
!Producer #1 wrote: a // Idle time: 1s
!Producer #0 wrote: b // Idle time: 2s
!Producer #2 wrote: c // Idle time: 2s
?Consumer #2 read:  a // Idle time: 2s
!Producer #1 wrote: d // Idle time: 1s
!Producer #0 wrote: e // Idle time: 1s
!Producer #2 wrote: f // Idle time: 1s
?Consumer #0 read:  b // Idle time: 3s
?Consumer #1 read:  c // Idle time: 3s
!Producer #1 wrote: g // Idle time: 2s
!Producer #0 wrote: h // Idle time: 2s
!Producer #2 wrote: i // Idle time: 2s
?Consumer #1 read:  d // Idle time: 2s
?Consumer #2 read:  e // Idle time: 4s
!Producer #1 wrote: j // Idle time: 2s
!Producer #0 wrote: k // Idle time: 2s
!Producer #2 wrote: l // Idle time: 2s
?Consumer #1 read:  f // Idle time: 2s
?Consumer #0 read:  g // Idle time: 5s
!Producer #2 wrote: m // Idle time: 1s
!Producer #1 wrote: n // Idle time: 2s
?Consumer #1 read:  h // Idle time: 1s
?Consumer #0 read:  i // Idle time: 1s
!Producer #0 wrote: o // Idle time: 2s
?Consumer #1 read:  j // Idle time: 1s
?Consumer #2 read:  k // Idle time: 4s
!Producer #1 wrote: p // Idle time: 2s
!Producer #2 wrote: q // Idle time: 2s
!Producer #0 wrote: r // Idle time: 2s
!Producer #2 wrote: s // Idle time: 2s
!Producer #1 wrote: t // Idle time: 2s
?Consumer #0 read:  l // Idle time: 3s
?Consumer #2 read:  m // Idle time: 3s
?Consumer #1 read:  n // Idle time: 4s
!Producer #0 wrote: u // Idle time: 2s
?Consumer #0 read:  o // Idle time: 1s
```

Рисунок 1 – Демонстрация работы программы. Максимальная задержка потребителя – 5 секунд, производителя – 2 секунды

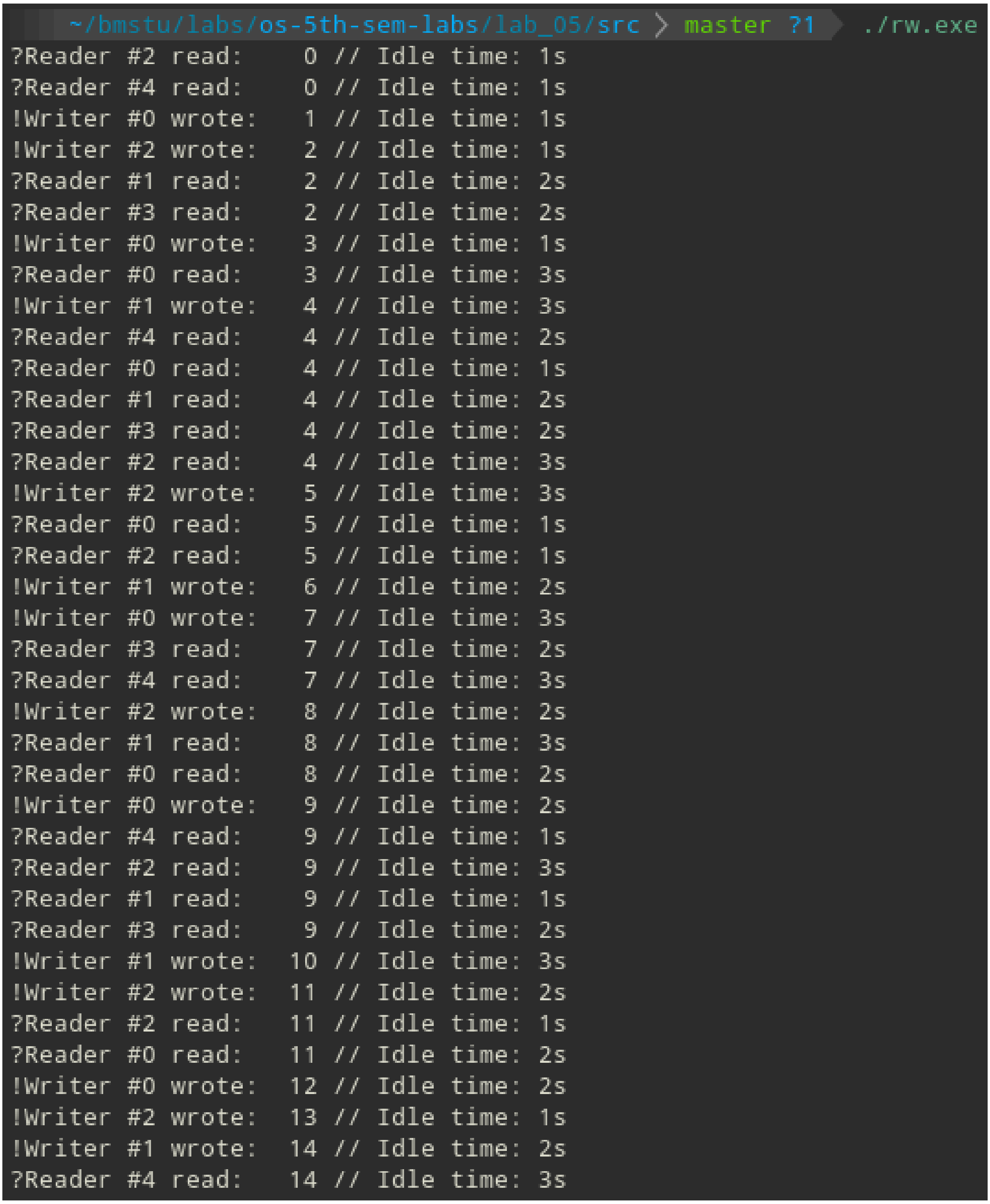
Задача «Читатели – писатели»

Листинг 2 – Реализация задачи «читатели – писатели»

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>
#include <wait.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <sys/stat.h>

#define MAX_SEMS 4
#define ITERS 20

#define READERS_COUNT 5
#define WRITERS_COUNT 3

#define ACTIVE_READERS 0
#define ACTIVE_WRITERS 1

#define WAITING_READERS 2
#define WAITING_WRITERS 3

#define MAX_RAND 3

struct sembuf START_READ[] = {
    {WAITING_READERS, 1, 0},
    {ACTIVE_WRITERS, 0, 0},
    {WAITING_WRITERS, 0, 0},
    {ACTIVE_READERS, 1, 0},
    {WAITING_READERS, -1, 0},
};

struct sembuf STOP_READ[]  = {
    {ACTIVE_READERS, -1, 0},
};

struct sembuf START_WRITE[]  = {
    {WAITING_WRITERS, 1, 0},
    {ACTIVE_READERS, 0, 0},
    {ACTIVE_WRITERS, 0, 0},
    {ACTIVE_WRITERS, 1, 0},
    {WAITING_WRITERS, -1, 0},
};

```

```c
struct sembuf STOP_WRITE[] = {
    {ACTIVE_WRITERS, -1, 0},
};

int start_read(int s_id)
{
    return semop(s_id, START_READ, 5) != -1;
}

int stop_read(int s_id)
{
    return semop(s_id, STOP_READ, 1) != -1;
}

int start_write(int s_id)
{
    return semop(s_id, START_WRITE, 5) != -1;
}

int stop_write(int s_id)
{
    return semop(s_id, STOP_WRITE, 1) != -1;
}

int rr_run(int *const shcntr, const int s_id, const int r_id)
{
    if (!shcntr)
    {
        return -1;
    }

    srand(time(NULL) + r_id);

    int stime;

    for (size_t i = 0; i < ITERS; ++i)
    {
        stime = rand() % MAX_RAND + 1;
        sleep(stime);

        if (!start_read(s_id))
        {
            perror("Reading␣start␣error.");

            exit(EXIT_FAILURE);
        }

        int val = *shcntr;
        printf("?Reader␣#%d␣read:␣␣%3d␣//␣Idle␣time:␣%ds\n", r_id, val, stime);

        if (!stop_read(s_id))
        {
            perror("Reading␣end␣error.");

            exit(EXIT_FAILURE);
        }
    }

    return EXIT_SUCCESS;
}

int wr_run(int *const shcntr, const int s_id, const int w_id)
{
    if (!shcntr)
    {
        return -1;
    }

    srand(time(NULL) + w_id + READERS_COUNT);

    int stime;

    for (size_t i = 0; i < ITERS; ++i)
    {
        stime = rand() % MAX_RAND + 1;
        sleep(stime);

        if (!start_write(s_id))
        {
            perror("Writing␣start␣error.");

            exit(EXIT_FAILURE);
        }

        int val = ++(*shcntr);
        printf("!Writer␣#%d␣wrote:␣%3d␣//␣Idle␣time:␣%ds\n", w_id, val, stime);

        if (!stop_write(s_id))
        {
            perror("Writing␣end␣error.");

            exit(EXIT_FAILURE);
        }
    }

    return EXIT_SUCCESS;
}

int main()
{
    setbuf(stdout, NULL);

    int fd = shmget(IPC_PRIVATE, sizeof(int), IPC_CREAT | S_IRWXU | S_IRWXG | S_IRWXO);
    if (fd == -1)
    {
        perror("shmget␣failed.");

        return EXIT_FAILURE;
    }

    int *shcntr = shmat(fd, 0, 0);
    if (shcntr == (void *)-1)
    {
        perror("shmat␣failed.");

        return EXIT_FAILURE;
    }

    int s_id = semget(IPC_PRIVATE, MAX_SEMS, IPC_CREAT | S_IRWXU | S_IRWXG | S_IRWXO);
    if (s_id == -1)
    {
        perror("semget␣failed.");

        return EXIT_FAILURE;
    }

    semctl(s_id, ACTIVE_READERS, SETVAL, 0);
    semctl(s_id, ACTIVE_WRITERS, SETVAL, 0);
    semctl(s_id, WAITING_WRITERS, SETVAL, 0);
    semctl(s_id, WAITING_READERS, SETVAL, 0);

    int chpid;
    for (size_t i = 0; i < READERS_COUNT; ++i)
    {
        switch ((chpid = fork()))
        {
        case -1:
            perror("Reader␣fork␣failed.");

            exit(EXIT_FAILURE);
```

```c
            break;
        case 0:
            rr_run(shcntr, s_id, i);

            return EXIT_SUCCESS;
        }
    }

    for (size_t i = 0; i < WRITERS_COUNT; ++i)
    {
        switch ((chpid = fork()))
        {
        case -1:
            perror("Writer␣fork␣failed.");

            exit(EXIT_FAILURE);
            break;
        case 0:
            wr_run(shcntr, s_id, i);

            return EXIT_SUCCESS;
        }
    }

    for (size_t i = 0; i < WRITERS_COUNT + READERS_COUNT; ++i)
    {
        int status;
        if (wait(&status) == -1)
        {
            perror("Child␣error.");

            exit(EXIT_FAILURE);
        }
        if (!WIFEXITED(status))
        {
            printf("Child␣process␣terminated␣abnormally\n");
        }
    }

    if (shmdt((void *)shcntr) == -1 ||
        shmctl(fd, IPC_RMID, NULL) == -1 ||
        semctl(s_id, IPC_RMID, 0) == -1)
    {

        perror("Exit␣error.");

        return EXIT_FAILURE;
    }

    return EXIT_SUCCESS;
}
```



```
~/bmstu/labs/os-5th-sem-labs/lab_05/src  master ?1  ./rw.exe
?Reader #2 read:    0 // Idle time: 1s
?Reader #4 read:    0 // Idle time: 1s
!Writer #0 wrote:   1 // Idle time: 1s
!Writer #2 wrote:   2 // Idle time: 1s
?Reader #1 read:    2 // Idle time: 2s
?Reader #3 read:    2 // Idle time: 2s
!Writer #0 wrote:   3 // Idle time: 1s
?Reader #0 read:    3 // Idle time: 3s
!Writer #1 wrote:   4 // Idle time: 3s
?Reader #4 read:    4 // Idle time: 2s
?Reader #0 read:    4 // Idle time: 1s
?Reader #1 read:    4 // Idle time: 2s
?Reader #3 read:    4 // Idle time: 2s
?Reader #2 read:    4 // Idle time: 3s
!Writer #2 wrote:   5 // Idle time: 3s
?Reader #0 read:    5 // Idle time: 1s
?Reader #2 read:    5 // Idle time: 1s
!Writer #1 wrote:   6 // Idle time: 2s
!Writer #0 wrote:   7 // Idle time: 3s
?Reader #3 read:    7 // Idle time: 2s
?Reader #4 read:    7 // Idle time: 3s
!Writer #2 wrote:   8 // Idle time: 2s
?Reader #1 read:    8 // Idle time: 3s
?Reader #0 read:    8 // Idle time: 2s
!Writer #0 wrote:   9 // Idle time: 2s
?Reader #4 read:    9 // Idle time: 1s
?Reader #2 read:    9 // Idle time: 3s
?Reader #1 read:    9 // Idle time: 1s
?Reader #3 read:    9 // Idle time: 2s
!Writer #1 wrote:  10 // Idle time: 3s
!Writer #2 wrote:  11 // Idle time: 2s
?Reader #2 read:   11 // Idle time: 1s
?Reader #0 read:   11 // Idle time: 2s
!Writer #0 wrote:  12 // Idle time: 2s
!Writer #2 wrote:  13 // Idle time: 1s
!Writer #1 wrote:  14 // Idle time: 2s
?Reader #4 read:   14 // Idle time: 3s
```

Рисунок 2 – Демонстрация работы программы. Максимальная задержка – 3 секунды