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»

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

Тема Взаимодействие параллельных процессов

Студент Якуба Д.В.

Группа ИУ7-53Б

Оценка (баллы) _____

Преподаватель Рязанова Н.Ю.

Москва, 2020 г

Задание 1.

Написать программу, реализующую задачу «Производство-потребление» по алгоритму Э. Дейкстры с тремя семафорами: двумя считающими и одним бинарным. В программе должно создаваться не менее 3х процессов - производителей и 3х процессов – потребителей. В программе надо обеспечить случайные задержки выполнения созданных процессов. В программе для взаимодействия производителей и потребителей буфер создается в разделяемом сегменте. Обратите внимание на то, чтобы не работать с одиночной переменной, а работать именно с буфером, состоящим их N ячеек по алгоритму. Производители в ячейки буфера записывают буквы алфавита по порядку. Потребители считывают символы из доступной ячейки. После считывания буквы из ячейки следующий потребитель может взять букву из следующей ячейки.

Листинг кода 1

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

#define SEM_AMOUNT 3
#define SEM_BIN 0
#define SEM_E 1
#define SEM_F 2
#define EMPTY_NUM 20
#define PRODUCE_NUM 6
#define CONSUME_NUM 6
#define CYCLES 3
#define SEM_ERROR 1
#define SEM_SET_ERR 2
#define SHM_ERROR 3
#define MEM_ERR 4
#define FORK_ERR 5
#define SEMOP_ERR 6

int *sharedMemoryPtr = NULL;
char *sharedCharMemoryPtr = NULL;

char *alphabet = "abcdefghijklmnopqrstuvwxyz";

struct sembuf prodStart[2] =
{
    {SEM_E, -1, 1},
    {SEM_BIN, -1, 1}
};
struct sembuf prodEnd[2] =
{
    {SEM_BIN, 1, 1},
    {SEM_F, 1, 1}
};
struct sembuf readStart[2] =
{
    {SEM_F, -1, 1},
```

```
    {SEM_BIN, -1, 1}
};
struct sembuf readEnd[2] =
{
    {SEM_BIN, 1, 1},
    {SEM_E, 1, 1}
};

void producer(int semID, int prodID)
{
    srand(time(NULL));
    sleep(rand() % 4);

    if (semop(semID, prodStart, 2) == -1)
    {
        perror("Producer semop error");
        exit(SEMOP_ERR);
    }

    sharedCharMemoryPtr[sharedMemoryPtr[0]] = alphabet[sharedMemoryPtr[0]];

    printf("<<---Producer[ID = %d]: wrote %c\n", prodID,
sharedCharMemoryPtr[sharedMemoryPtr[0]]);
    sharedMemoryPtr[0]++;

    if (semop(semID, prodEnd, 2) == -1)
    {
        perror("Producer semop error");
        exit(SEMOP_ERR);
    }
}

void consumer(int semID, int consID)
{
    srand(time(NULL));
    sleep(rand() % 4);
    if (semop(semID, readStart, 2) == -1)
    {
        perror("Consumer semop error");
        exit(SEMOP_ERR);
    }

    printf("->>Consumer[ID = %d]: read %c\n", consID,
sharedCharMemoryPtr[sharedMemoryPtr[1]]);
    sharedMemoryPtr[1]++;

    if (semop(semID, readEnd, 2) == -1)
    {
        perror("Consumer semop error");
        exit(SEMOP_ERR);
    }
}

int main()
{
    int semID = semget(IPC_PRIVATE, SEM_AMOUNT, IPC_CREAT | S_IRUSR | S_IWUSR | S_IRGRP |
S_IROTH);
    if (semID == -1)
    {
        perror("Semaphore creation error.");
        exit(SEM_ERROR);
    }

    if (semctl(semID, SEM_BIN, SETVAL, 1) == -1 ||
        semctl(semID, SEM_E, SETVAL, EMPTY_NUM) == -1 ||
        semctl(semID, SEM_F, SETVAL, 0) == -1)
    {
        perror("Semaphore set error.");
```

```
        exit(SEM_SET_ERR);
    }

    int shmID = shmget(IPC_PRIVATE, 2 * sizeof(int) + EMPTY_NUM * sizeof(char), S_IRUSR |
S_IWUSR | S_IRGRP | S_IROTH);
    if (shmID == -1)
    {
        perror("Shared memory creation error.");
        exit(SHM_ERROR);
    }

    sharedMemoryPtr = shmat(shmID, 0, 0);

    if (*sharedMemoryPtr == -1)
    {
        perror("Memory all error.");
        exit(MEM_ERR);
    }
    sharedCharMemoryPtr = (char *)(sharedMemoryPtr + 2 * sizeof(int));

    pid_t childID = -1;

    for (int i = 0; i < CYCLES; ++i)
    {
        if ((childID = fork()) == -1)
        {
            perror("Producer fork error");
            exit(FORK_ERR);
        }
        else if (childID == 0)
        {
            for (int j = 0; j < PRODUCE_NUM; j++)
                producer(semID, i);
            exit(0);
        }
        if ((childID = fork()) == -1)
        {
            perror("Consumer fork error");
            exit(FORK_ERR);
        }
        else if (childID == 0)
        {
            for (int j = 0; j < CONSUME_NUM; j++)
                consumer(semID, i);
            exit(0);
        }
    }

    int status;
    for (int i = 0; i < CONSUME_NUM + PRODUCE_NUM; i++)
        wait(&status);

    if (shmdt(sharedMemoryPtr) == -1)
    {
        perror("SHMDT error");
        exit(MEM_ERR);
    }
    if (shmctl(shmID, IPC_RMID, NULL) == -1)
    {
        perror("SHMCTL error");
        exit(MEM_ERR);
    }
    sharedCharMemoryPtr = NULL;

    printf("\n\nThis is the end of the task\n");
    exit(0);
}
```

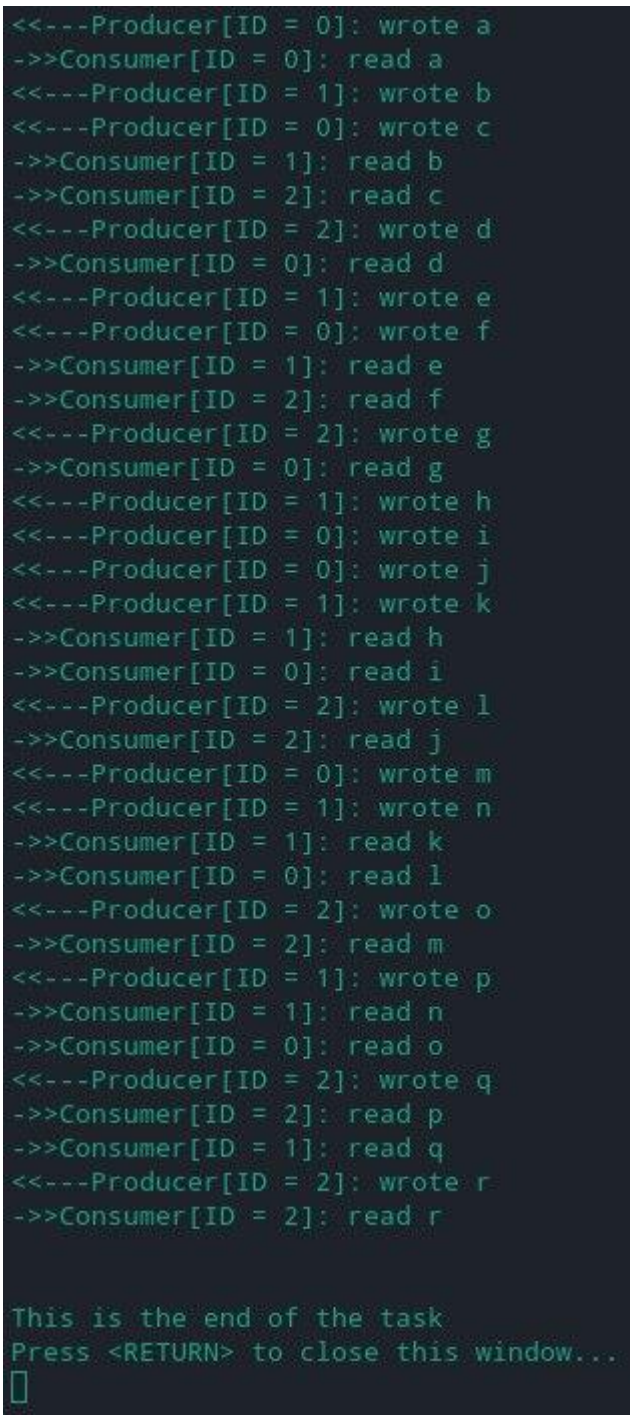

Рисунок 1, Демонстрация работы программы (задержки производителей от 0 до 4, задержки потребителей от 0 до 4)

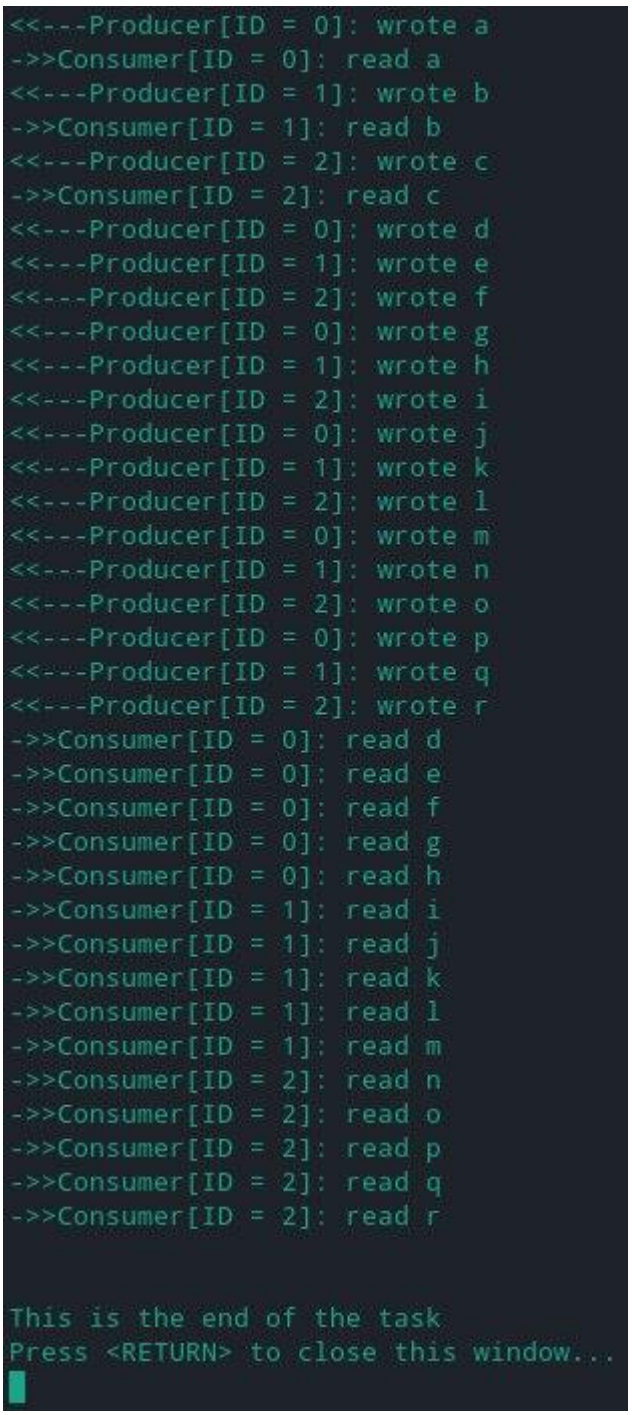

Рисунок 2, Демонстрация работы программы (задержки производителей от 0 до 3, задержки потребителей от 0 до 4)

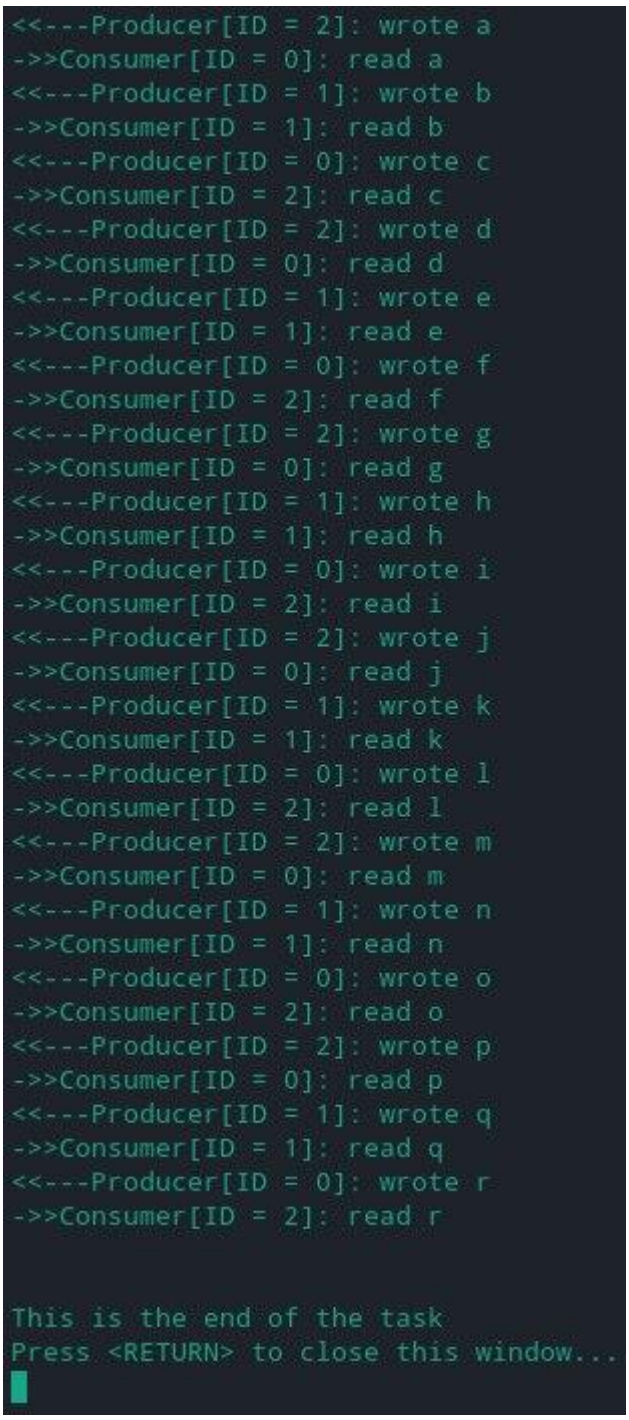

Рисунок 3, Демонстрация работы программы (задержки производителей от 0 до 4, задержки потребителей от 0 до 2)

Задание 2.

Написать программу, реализующую задачу «Читатели – писатели» по монитору Хоара с четырьмя функциями: Начать_чтение, Закончить_чтение, Начать_запись, Закончить_запись. В программе всеми процессами разделяется одно единственное значение в разделяемой памяти. Писатели ее только инкрементируют, читатели могут только читать значение.

Листинг кода 2

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

#define SEM_AMOUNT 4

#define WRITERS_AMOUNT 3
#define READERS_AMOUNT 5

#define ACT_READER 0
#define ACT_WRITER 1
#define BIN_ACT_WRITER 2
#define WAIT_WRITER 3

#define SEM_ERROR 1
#define SEM_SET_ERR 2
#define SHM_ERR 3
#define MEM_ERR 4
#define FORK_ERR 5
#define SEMOP_ERR 6

struct sembuf startRead[] =
{
    { WAIT_WRITER, 0, 1 },
    { ACT_WRITER, 0, 1 },
    { ACT_READER, 1, 1 }
};

struct sembuf stopRead[] =
{
    { ACT_READER, -1, 1 }
};

struct sembuf startWrite[] =
{
    { WAIT_WRITER, 1, 1 },
    { ACT_READER, 0, 1 },
    { BIN_ACT_WRITER, -1, 1 },
    { ACT_WRITER, 1, 1 },
    { WAIT_WRITER, -1, 1 }
};

struct sembuf stopWrite[] =
{
    { ACT_WRITER, -1, 1 },
    { BIN_ACT_WRITER, 1, 1 }
};

int *sharedMemoryPtr = NULL;
```

```
void writer(int semID, int writerID)
{
    if (semop(semID, startWrite, 5) == -1)
    {
        perror("Semop error");
        exit(SEMOP_ERR);
    }

    (*sharedMemoryPtr)++;
    printf("<<---Writer[ID = %d]: write value %d\n", writerID, *sharedMemoryPtr);

    if (semop(semID, stopWrite, 2) == -1)
    {
        perror("Writer semop error");
        exit(SEMOP_ERR);
    }

    sleep(1);
}

void reader(int semID, int readerID)
{
    if (semop(semID, startRead, 3) == -1)
    {
        perror("Semop error");
        exit(SEMOP_ERR);
    }

    printf("->>Reader[ID = %d]: reads value %d\n", readerID, *sharedMemoryPtr);

    if (semop(semID, stopRead, 1) == -1)
    {
        perror("Writer semop error");
        exit(SEMOP_ERR);
    }

    sleep(1);
}

int main()
{
    int semID = semget(IPC_PRIVATE, SEM_AMOUNT, IPC_CREAT | S_IRUSR | S_IWUSR | S_IRGRP |
S_IROTH);
    if (semID == -1)
    {
        perror("Semaphore creation error.");
        exit(SEM_ERROR);
    }

    if (semctl(semID, 2, SETVAL, 1) == -1)
    {
        perror("Semaphore set error.");
        exit(SEM_SET_ERR);
    }

    int shmID = shmget(IPC_PRIVATE, sizeof(int), S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH);
    if (shmID == -1)
    {
        perror("Shared memory creation error.");
        exit(SHM_ERR);
    }

    sharedMemoryPtr = shmat(shmID, 0, 0);
    if (*sharedMemoryPtr == -1)
    {
        perror("Memory all error.");
        exit(MEM_ERR);
```

```
    }

    pid_t childID = -1;
    for (int i = 0; i < WRITERS_AMOUNT; i++)
    {
        if ((childID = fork()) == -1)
        {
            perror("Write fork error");
            exit(FORK_ERR);
        }
        else if (childID == 0)
        {
            for (;;)
                writer(semID, i);
            exit(0);
        }
    }

    for (int i = 0; i < READERS_AMOUNT; i++)
    {
        if ((childID = fork()) == -1)
        {
            perror("Reader fork error");
            exit(FORK_ERR);
        }
        else if (childID == 0)
        {
            for (;;)
                reader(semID, i);
            exit(0);
        }
    }

    int status;
    for (int i = 0; i < WRITERS_AMOUNT + READERS_AMOUNT; i++)
        wait(&status);

    if (shmdt(sharedMemoryPtr) == -1)
    {
        perror("SHMDT error");
        exit(MEM_ERR);
    }

    if (shmctl(shmID, IPC_RMID, NULL) == -1)
    {
        perror("SHMCTL error");
        exit(MEM_ERR);
    }
}
```

Пример работы программы предоставлен на следующей странице (Рисунок 4).

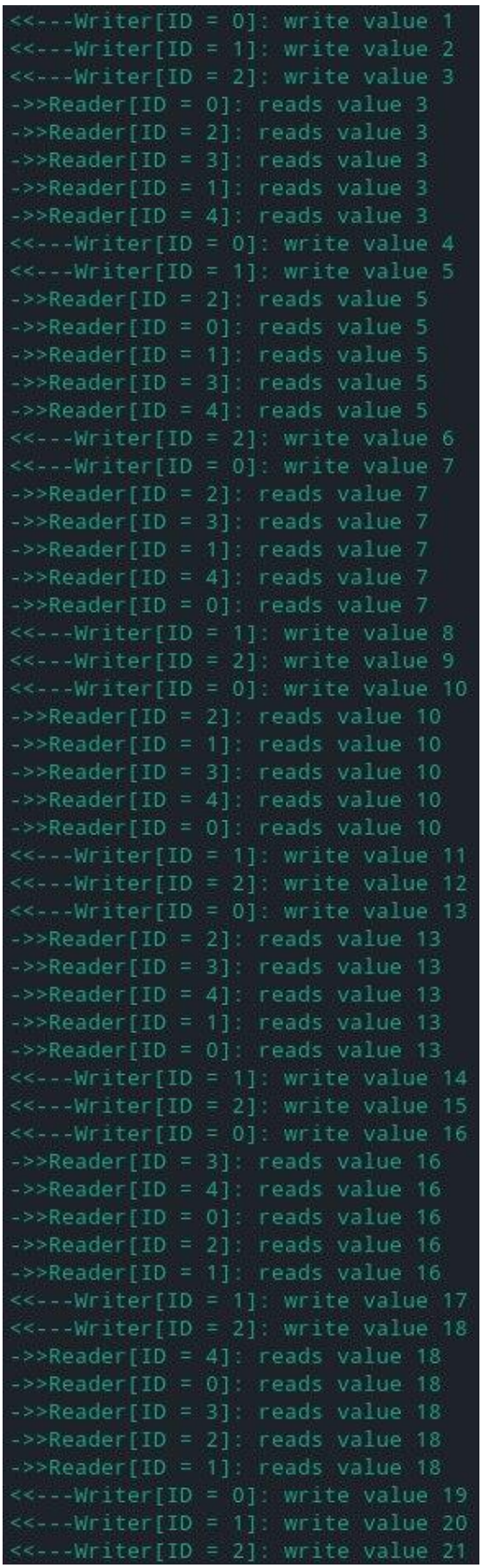

Рисунок 4, Демонстрация работы программы