|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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**

***к лабораторной работе №5 по курсу «Операционные системы»***

***по теме «Взаимодействие параллельных процессов»***

|  |  |
| --- | --- |
| **Студент: Русинова Д. Э.**  **Группа: ИУ7-55б**  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель: Рязанова Н. Ю.** |  |

Москва.

2020 г.

1. Реализация задачи «Производство-потребление»: 3 процесса производителя, 3 процесса потребителя. 3 семафора: 2 считающих и 1 бинарный.

#include <sys/sem.h> // struct sembuf

#include <sys/stat.h> // IPC\_CREAT | S\_IRWXU | S\_IRWXG | S\_IRWXO

#include <sys/shm.h> // разделяемая память

#include <sys/wait.h> // wait(int \*status)

#include <stdio.h> // printf

#include <stdlib.h> // rand()

#include <unistd.h> // fork

const int FLAGS = IPC\_CREAT | S\_IRWXU | S\_IRWXG | S\_IRWXO;

#define SEM\_ID 123  // идентификатор для создания набора семафоров

#define SHM\_ID 123  // идентификатор для создание сегмента разделяемой памяти

#define PRODS 3 // кол-во производитлей

#define CONS 3  // кол-во потребителей

// идентификаторы семафоров в наборе

#define FULL\_SEM\_NUM  0

#define EMPTY\_SEM\_NUM 1

#define BIN\_SEM\_NUM   2

// буфер организован по принципу FIFO (цикл. очередь)

char\* buf;

char\* letter; // текущая буква алфавита

const int bufsize = 10;

int\* produce\_pos; // индекс последней свободной ячейки - туда пишем

int\* consume\_pos; // индекс первой занятой ячейки - оттуда читаем

int\* stop\_consuming; // флаг, по установке которого в 1,

// процессы потребители завершаются

                  // в программе этот флаг устанавливается,

// когда процесс протребитель обнаруживает букву z

// в semop передается массив sembuf - операций над семафорами набора

// над семафорами определены следующие операции:

// перед входом в крит секцию производителя

struct sembuf producer\_P[2] = {

    {EMPTY\_SEM\_NUM, -1, 0},

    {BIN\_SEM\_NUM, -1, 0}  };

// после выхода из крит секции производителя

struct sembuf producer\_V[2] = {

    {BIN\_SEM\_NUM, 1, 0},

    {FULL\_SEM\_NUM, 1, 0}  };

// перед входом в крит секцию потребителя

struct sembuf consumer\_P[2] = {

    {FULL\_SEM\_NUM, -1, 0},

    {BIN\_SEM\_NUM, -1, 0}  };

// после выхода из крит секции потребителя

struct sembuf consumer\_V[2] = {

    {BIN\_SEM\_NUM, 1, 0},

    {EMPTY\_SEM\_NUM, 1, 0} };

// процесс-потребитель

// на каждой итерации выбирает из буфера единичный объект

int consume(int i, int sem\_fd)

{

    while(!(\*stop\_consuming))

    {

        // ожидание заполнения хотя бы одной ячейки буфера и

        // ожидание выхода из крит секции другого процесса

        semop(sem\_fd, consumer\_P, 2);

        // потребление единичного объекта

        char let = buf[\*consume\_pos];

        printf("Consumer № %d consumed object on position %d: %c\n",

i, \*consume\_pos, let);

        (\*consume\_pos) = ((\*consume\_pos) + 1) % bufsize;

        // освобождение критической секции и

        // инкремент количества пустых ячеек

        semop(sem\_fd, consumer\_V, 2);

        sleep(rand() % 3);

        // как только хотя бы один из процессов обнаруживает букву z,

        // все процессы потребители завершаются

        if (let == 'z')

            (\*stop\_consuming) = 1;

    }

    return 0;

}

// процесс-производитель

// на каждой итерации производит единичный объект и помещает его в буфер

int produce(int i, int sem\_fd)

{

    while(\*letter <= 'z')

    {

        // ожидание, пока освободится хотя бы одна ячейка буфера и

        // ожидание выхода из крит секции другого процесса

        semop(sem\_fd, producer\_P, 2);

        // производство единичного объекта

        buf[\*produce\_pos] = \*letter;

        printf("\t\t\t\t\tProducer № %d produced object on position %d: %c\n", i, \*produce\_pos, buf[\*produce\_pos]);

        (\*produce\_pos) = ((\*produce\_pos) + 1) % bufsize;

        (\*letter)++;

        // освобождение критической секции и

        // инкремент количества заполненных ячеек

        semop(sem\_fd, producer\_V, 2);

        sleep(rand() % 6);

    }

    return 0;

}

int create\_shared\_memory(int \*shm\_fd)

{

    // shmget() ; создает разделяемый сегмент

    size\_t size = 3 \* sizeof(int) + sizeof(char) \* (bufsize + 1);

    \*shm\_fd = shmget(SHM\_ID, size, FLAGS);

    if (\*shm\_fd == -1)

    {

        printf("Error in shmget\n");

        return -1;

    }

    // shmat() attach получение указателя на разделяемый сегмен

    // On success, shmat() returns the address of the attached shared memory

    // segment; on error, (void \*) -1 is returned

    produce\_pos = (int\*) shmat(\*shm\_fd, 0, 0);

    if (produce\_pos == (int\*) -1)

    {

        printf("Error in shmat\n");

        return -1;

    }

    else

    {

        consume\_pos = (produce\_pos + sizeof(int));

        stop\_consuming = (consume\_pos + sizeof(int));

        letter = (char \*)(stop\_consuming + sizeof(int));

        buf = letter + sizeof(char);

        \*letter = 'a';

        \*produce\_pos = 0;

        \*consume\_pos = 0;

        \*stop\_consuming = 0;

    }

    return 0;

}

int create\_semaphores(int \*sem\_fd)

{

    // semget(); создаёт набор семафоров

    if ((\*sem\_fd = semget(SEM\_ID, 3, FLAGS)) == -1)

    {

        printf("Error in semget\n");

        return 1;

    }

    //  int semctl(int semid, int semnum, int cmd, ...);

    // performs the control operation specified by cmd on the semnum-th

    // the semaphore set identified by semid

    semctl(\*sem\_fd, 0, SETVAL, 0);

    semctl(\*sem\_fd, 1, SETVAL, bufsize);

    semctl(\*sem\_fd, 2, SETVAL, 1);

    return 0;

}

int main()

{

    // создание и присоединение сегмента разделяемой памяти

    int shm\_fd;

    if (create\_shared\_memory(&shm\_fd) == -1)

        return -1;

    // создание набора семафоров

    int sem\_fd;

    if (create\_semaphores(&sem\_fd) == -1)

        return -1;

    // создание процессов производителей

    for (int i = 0; i < PRODS; i++)

    {

        int pid;

        if ((pid = fork()) == -1) {

            printf("Can't fork producer");

            exit(1);

        }

        if (pid == 0)

        {

            // код процесса потомка

            produce(i+1, sem\_fd);

            return 0;

        }

    }

// создание процессов потребителй

    for (int i = 0; i < CONS; i++)

    {

        int pid;

        if ((pid = fork()) == -1) {

            printf("Can't fork consumer");

            exit(1);

        }

        if (pid == 0)

        {

            // код процесса потомка

            consume(i+1, sem\_fd);

            return 0;

        }

    }

    for (int i = 0; i < PRODS + CONS; i++)

    {

        // процесс предок дожидается завершения процессов потомков

        int status;

        wait(&status);

    }

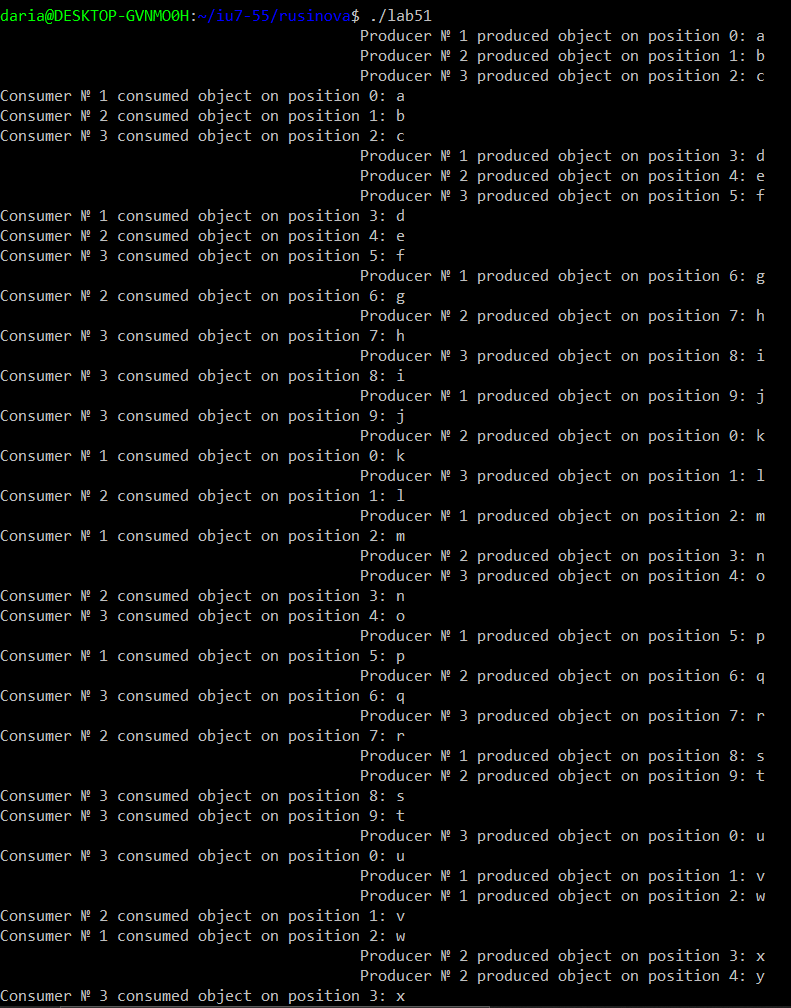
shmdt(produce\_pos);  
 shmctl(shm\_fd, IPC\_RMID, 0);

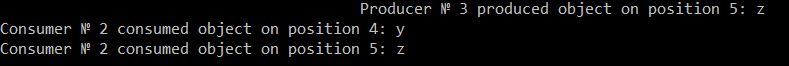
    semctl(sem\_fd, 0, IPC\_RMID);

    return 0;

}

Демонстрация работы программы приведена на следующем рисунке.





1. Реализация задачи «Читатели – писатели»: 3 процесса писателя, 5 процессов потребителей. 4 семафора: 3 считающих и 1 бинарный.

#include <sys/sem.h> // struct sembuf

#include <sys/stat.h> // IPC\_CREAT | S\_IRWXU | S\_IRWXG | S\_IRWXO

#include <sys/shm.h> // разделяемая память

#include <sys/wait.h> // wait(int \*status)

#include <stdio.h> // printf

#include <stdlib.h> // rand()

#include <unistd.h> // fork

#define COUNT 14  // как только значение ресурса достигает 14,

                  // завершаем работу программы

#define WRITERS 3

#define READERS 5

const int FLAGS = IPC\_CREAT | S\_IRWXU | S\_IRWXG | S\_IRWXO;

#define SEM\_ID 321  // идентификатор для создания набора семафоров

#define SHM\_ID 321  // идентификатор для создание сегмента разделяемой памяти

// идентификаторы семафоров в наборе

#define ACTIVE\_READERS\_SEM 0

#define ACTIVE\_WRITER\_SEM 1

#define WAITING\_READERS\_SEM 2

#define WAITING\_WRITERS\_SEM 3

// до входа писателя в крит секцию

struct sembuf start\_write[5] = {

    { WAITING\_WRITERS\_SEM,  1, 0 },

    { ACTIVE\_READERS\_SEM,   0, 0 },

    { ACTIVE\_WRITER\_SEM,    0, 0 },

    { ACTIVE\_WRITER\_SEM,    1, 0 },

    { WAITING\_WRITERS\_SEM, -1, 0 }

};

// после выхода писателя из крит секции

struct sembuf stop\_write[1] = {

    { ACTIVE\_WRITER\_SEM, -1, 0 }

};

// до входа читателя в крит секцию

struct sembuf start\_read[5] = {

    { WAITING\_READERS\_SEM,  1, 0 },

    { WAITING\_WRITERS\_SEM,  0, 0 },

    { ACTIVE\_WRITER\_SEM,    0, 0 },

    { ACTIVE\_READERS\_SEM,   1, 0 },

    { WAITING\_READERS\_SEM, -1, 0 }

};

// после выхода читателя из крит секции

struct sembuf stop\_read[1] = {

    { ACTIVE\_READERS\_SEM, -1, 0 }

};

// РЕСУРС

int\* resource;

void start\_reader(int sem\_fd) { semop(sem\_fd, start\_read, 5); }

void stop\_reader(int sem\_fd) { semop(sem\_fd, stop\_read, 1); }

void start\_writer(int sem\_fd) { semop(sem\_fd, start\_write, 5); }

void stop\_writer(int sem\_fd) { semop(sem\_fd, stop\_write, 1); }

// процесс-писатель

void writer(int i, int sem\_fd)

{

    while(\*resource < COUNT)

    {

        start\_writer(sem\_fd);

        // изменяем ресурс

        (\*resource)++;

        printf("\t\tWriter № %d increased resource up to %d\n", i, \*resource);

        stop\_writer(sem\_fd);

        sleep(rand() % 4);

    }

}

// процесс-читатель

void reader(int i, int sem\_fd)

{

    int stop = 0;

    while(!stop)

    {

        start\_reader(sem\_fd);

        // читаем ресурс

        printf("Reader № %d read resource: %d\n", i, \*resource);

        if (\*resource == COUNT)

            stop = 1;

        stop\_reader(sem\_fd);

        sleep(rand() % 4);

    }

}

int create\_shared\_memory(int \*shm\_fd)

{

    // shmget() ; создает разделяемый сегмент

    \*shm\_fd = shmget(SHM\_ID, sizeof(int), FLAGS);

    if (\*shm\_fd == -1)

    {

        printf("Error in shmget\n");

        return -1;

    }

    // shmat() attach получение указателя на разделяемый сегмен

    resource = (int\*) shmat(\*shm\_fd, 0, 0);

    if (resource == (int\*)-1)

    {

        printf("Error in shmat\n");

        return -1;

    }

    (\*resource) = 0;

    return 0;

}

int create\_semaphores(int \*sem\_fd)

{

    // semget(); создаёт набор семафоров

    if ((\*sem\_fd = semget(SEM\_ID, 4, FLAGS)) == -1)

    {

        printf("Error in semget\n");

        return 1;

    }

    semctl(\*sem\_fd, ACTIVE\_READERS\_SEM, SETVAL, 0);

    semctl(\*sem\_fd, ACTIVE\_WRITER\_SEM, SETVAL, 0);

    semctl(\*sem\_fd, WAITING\_READERS\_SEM, SETVAL, 0);

    semctl(\*sem\_fd, WAITING\_WRITERS\_SEM, SETVAL, 0);

    return 0;

}

int main()

{

    // создание и присоединение сегмента разделяемой памяти.

    int shm\_fd;

    if (create\_shared\_memory(&shm\_fd) == -1)

        return -1;

    // создание набора семафоров

    int sem\_fd;

    if (create\_semaphores(&sem\_fd) == -1)

        return -1;

    for (int i = 0; i < WRITERS; i++)

    {

        int pid;

        if ((pid = fork()) == -1) {

            printf("Can't fork writer");

            exit(1);

        }

        if (pid == 0)

        {

            // код процесса потомка

            writer(i+1, sem\_fd);

            return 0;

        }

    }

    for (int i = 0; i < READERS; i++)

    {

        int pid;

        if ((pid = fork()) == -1) {

            printf("Can't fork reader");

            exit(1);

        }

        if (pid == 0)

        {

            // код процесса потомка

            reader(i+1, sem\_fd);

            return 0;

        }

    }

    for (int i = 0; i < WRITERS + READERS; i++)

    {

        // процесс предок дожидается завершения процессов потомков

        int status;

        wait(&status);

    }

shmdt(resource);  
 shmctl(shm\_fd, IPC\_RMID, 0);

    semctl(sem\_fd, 0, IPC\_RMID);

    return 0;

}

Демонстрация работы программы приведена на следующем рисунке.

