|  |  |
| --- | --- |
| 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»\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ДИСЦИПЛИНА «Операционные системы»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Лабораторная работа № 5**

|  |  |
| --- | --- |
| **Тема** «Взаимодействие параллельных процессов»  **Студентка** Ловцова К.А.  **Группа** ИУ7-51Б  **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 3 изводителей 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.

**Код программы**

#include <sys/ipc.h>

#include <sys/sem.h>

#include <sys/shm.h>

#include <sys/stat.h>

#include <sys/wait.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#define CNT\_BUF 5

#define VALUES 10

#define SB 0

#define SE 1

#define SF 2

struct sembuf P\_start[2] = { {SE, -1, 0}, {SF, -1, 0} };

struct sembuf P\_stop[2] = { {SB, 1, 0}, {SF, 1, 0} };

struct sembuf C\_start[2] = { {SB, -1, 0}, {SF, -1, 0} };

struct sembuf C\_stop[2] = { {SE, 1, 0}, {SF, 1, 0} };

int\* shared\_buffer;

int\* shared\_pos\_cons;

int\* shared\_pos\_prod;

void producer(int isem\_descr, pid\_t pid)

{

    for (int i = 0; i < VALUES; i++)

    {

        printf("PID: %d\t", pid);

        sleep(rand() % 5);

        if (semop(isem\_descr, P\_start, 2) == -1)

        {

            perror("semop error");

            exit(1);

        }

        shared\_buffer[\*shared\_pos\_prod] = \*shared\_pos\_prod + 'a';

        printf("Producer #%d: %c\n", i, shared\_buffer[\*shared\_pos\_prod]);

        (\*shared\_pos\_prod)++;

        if (semop(isem\_descr, P\_stop, 2) == -1)

        {

            perror("semop error");

            exit(1);

        }

    }

}

void consumer(int isem\_descr, pid\_t pid)

{

    for (int i = 0; i < VALUES; i++)

    {

        printf("PID: %d\t", pid);

        sleep(rand() % 2);

        if (semop(isem\_descr, C\_start, 2) == -1)

        {

            perror("semop error");

            exit(1);

        }

        printf("Consumer #%d: %c\n", i, shared\_buffer[\*shared\_pos\_cons]);

        (\*shared\_pos\_cons)++;

        if (semop(isem\_descr, C\_stop, 2) == -1)

        {

            perror("semop error");

            exit(1);

        }

    }

}

int main()

{

    int perms = S\_IRWXU | S\_IRWXG | S\_IRWXO;

    int fd = shmget(IPC\_PRIVATE, (CNT\_BUF + 1) \* sizeof(char), IPC\_CREAT | perms);

    if (fd == -1)

    {

        perror("shmget error");

        exit(1);

    }

    shared\_pos\_prod = shmat(fd, 0, 0);

    if (\*shared\_pos\_prod == -1)

    {

        perror("shmat error");

        exit(1);

    }

    shared\_buffer = shared\_pos\_prod + 2 \* sizeof(char);

    shared\_pos\_cons = shared\_pos\_prod + sizeof(char);

    (\*shared\_pos\_prod) = 0;

    (\*shared\_pos\_cons) = 0;

    int isem\_descr = semget(IPC\_PRIVATE, 3, IPC\_CREAT | perms);

    if (isem\_descr == -1)

    {

        perror("semget error");

        exit(1);

    }

    int ctlf = semctl(isem\_descr, SB, SETVAL, 0);

    int ctle = semctl(isem\_descr, SE, SETVAL, CNT\_BUF);

    int ctlb = semctl(isem\_descr, SF, SETVAL, 1);

    if (ctlf == -1 || ctle == -1 || ctlb == -1)

    {

        perror("semctl error");

        exit(1);

    }

    pid\_t childpids[CNT\_BUF];

    childpids[0] = fork();

    if (childpids[0]  == -1)

    {

        perror("Can`t fork.\n");

        exit(1);

    }

    else if (childpids[0] == 0)

    {

        producer(isem\_descr, getpid());

        return 0;

    }

    else

    {

        childpids[1] = fork();

        if (childpids[1] == -1)

        {

            perror("Can`t fork.\n");

            exit(1);

        }

        else if (childpids[1] == 0)

        {

            producer(isem\_descr, getpid());

            return 0;

        }

        else

        {

            childpids[2] = fork();

            if (childpids[2] == -1)

            {

                perror("Can`t fork.\n");

                exit(1);

            }

            else if (childpids[2] == 0)

            {

                producer(isem\_descr, getpid());

                return 0;

            }

            else

            {

                childpids[3] = fork();

                if (childpids[3] == -1)

                {

                    perror("Can`t fork.\n");

                    exit(1);

                }

                else if (childpids[3] == 0)

                {

                    consumer(isem\_descr, getpid());

                    return 0;

                }

                else

                {

                   childpids[4] = fork();

                    if (childpids[4] == -1)

                    {

                        perror("Can`t fork.\n");

                        exit(1);

                    }

                    else if (childpids[4] == 0)

                    {

                        consumer(isem\_descr, getpid());

                        return 0;

                    }

                    else

                    {

                        consumer(isem\_descr, getpid());

                        int status;

                        for (int i = 0; i < CNT\_BUF; i++)

                            wait(&status);

                        if (shmdt(shared\_pos\_prod) == -1)

                        {

                            perror("shmdt error");

                            exit(1);

                        }

                        return 0;

                    }

                }

            }

        }

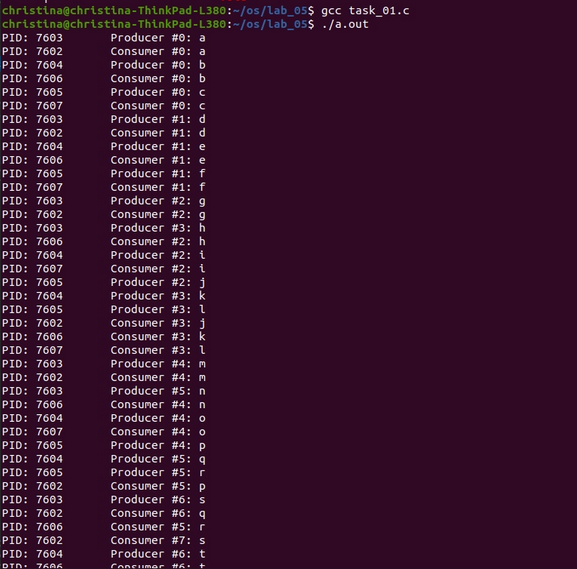
    }

    return 0;

}

**Пример работы программы**

Пример работы первой программы представлен на рисунках 1 и 2 ниже.



*Рисунок 1. – Пример работы первой программы.*



*Рисунок 2. – Пример работы первой программы.*

**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.

**Код программы**

#include <sys/ipc.h>

#include <sys/sem.h>

#include <sys/shm.h>

#include <sys/stat.h>

#include <sys/wait.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include <signal.h>

#include <unistd.h>

#define AR 0

#define AW 1

#define BW 2

#define WW 3

#define SEM\_CNT(a) sizeof(a)/sizeof(struct sembuf)

const int LOOP\_CNT = 10;

const int PERM = S\_IRWXU | S\_IRWXG | S\_IRWXO;

struct sembuf start\_write[] = {{WW, 1, 0}, {AR, 0, 0}, {BW, -1, 0}, {AW, 1, 0}, {WW, -1, 0}};

struct sembuf stop\_write[] = {{AW, -1, 0}, {BW, 1, 0}};

void writer(int semid, int\* shm, int num)

{

    int value = 0;

    while (value < LOOP\_CNT)

    {

        if (semop(semid, start\_write, SEM\_CNT(start\_write)))

        {

            perror("semop error");

            exit(1);

        }

        (\*shm)++;

        printf("PID: %d\tWriter #%d: %d\n", getpid(), num, \*shm);

        if (semop(semid, stop\_write, SEM\_CNT(stop\_write)))

        {

            perror("semop error");

            exit(1);

        }

        value++;

        sleep(rand() % 5);

    }

}

struct sembuf start\_read[] = {{WW, 0, 0}, {AW, 0, 0}, {AR, 1, 0}};

struct sembuf stop\_read[] = {{AR, -1, 0}};

void reader(int semid, int\* shm, int num)

{

    int value = 0;

    while (value < LOOP\_CNT)

    {

        if (semop(semid, start\_read, SEM\_CNT(start\_read)))

        {

            perror("semop error");

            exit(1);

        }

        printf("PID: %d\tReader #%d: %d\n", getpid(), num, \*shm);

        if (semop(semid, stop\_read, SEM\_CNT(stop\_read)))

        {

            perror("semop error");

            exit(1);

        }

        value++;

        sleep(rand() % 5);

    }

}

int main()

{

    int shm\_id;

    if ((shm\_id = shmget(IPC\_PRIVATE, 4, IPC\_CREAT | PERM)) == -1)

    {

        perror("shmget error");

        exit(1);

    }

    int\* shm\_buf = shmat(shm\_id, 0, 0);

    if (shm\_buf == -1)

    {

        perror("shmat error");

        exit(1);

    }

    (\*shm\_buf) = 0;

    int sem\_id;

    if ((sem\_id = semget(IPC\_PRIVATE, 4, IPC\_CREAT | PERM)) == -1)

    {

        perror("semget error");

        exit(1);

    }

    int ctrl = semctl(sem\_id, BW, SETVAL, 1);

    if (ctrl == -1)

    {

        perror("semctl error");

        exit(1);

    }

    pid\_t pid;

    for (int i = 0; i < 3 && pid != 0; i++)

    {

        pid = fork();

        if (pid == -1)

        {

            perror("Can`t fork.\n");

            exit(1);

        }

        if (pid == 0)

        {

            writer(sem\_id, shm\_buf, i);

        }

    }

    for (int i = 0; i < 5 && pid != 0; i++)

    {

        pid = fork();

        if (pid == -1)

        {

            perror("Can`t fork.\n");

            exit(1);

        }

        if (pid == 0)

        {

            reader(sem\_id, shm\_buf, i);

        }

    }

    if (pid != 0)

    {

      int\* status;

      for (int i = 0; i < 8; ++i)

      {

          wait(status);

      }

      if (shmdt(shm\_buf) == -1)

      {

          perror("shmdt error");

          exit(1);

      }

      if (shmctl(shm\_id, IPC\_RMID, NULL) == -1)

      {

          perror("shmctl error");

          exit(1);

      }

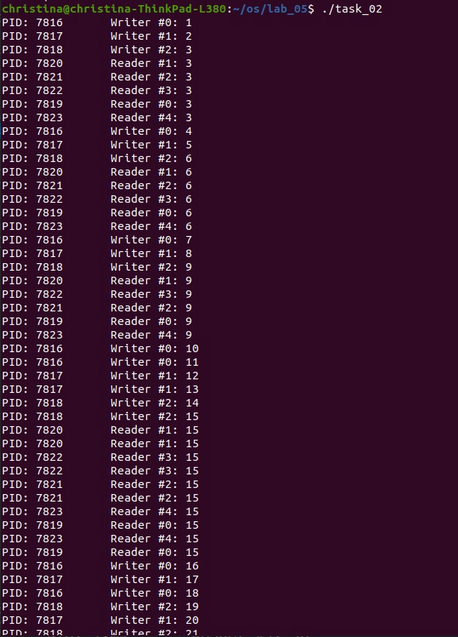
    }

    return 0;

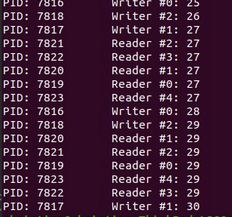
}

**Пример работы программы**

Пример работы второй программы представлен на рисунках 3 и 4 ниже.



*Рисунок 3. – Пример работы второй программы.*



*Рисунок 4. – Пример работы второй программы.*