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»

Студент Шиленков А. А.

Группа ИУ7-55Б

Оценка (баллы) _____

Преподаватель Рязанова Н.Ю.

Москва.
2020 г.

Цели и задачи

В лабораторной работе исследуются вопросы и формируются навыки использования в приложения таких средств меж процессного взаимодействия, как семафоры и разделяемая память. Работа выполняется на примере двух задач, характерных для взаимодействия асинхронных параллельных процессов: «производство-потребление» и «читатели-писатели».

В программах осуществляется консольный вывод, т.е. никакого интерфейса не нужно. Работающая программа должна выводить на экран какой процесс что записал, какой процесс что считал.

Задание 1

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

Код программы:

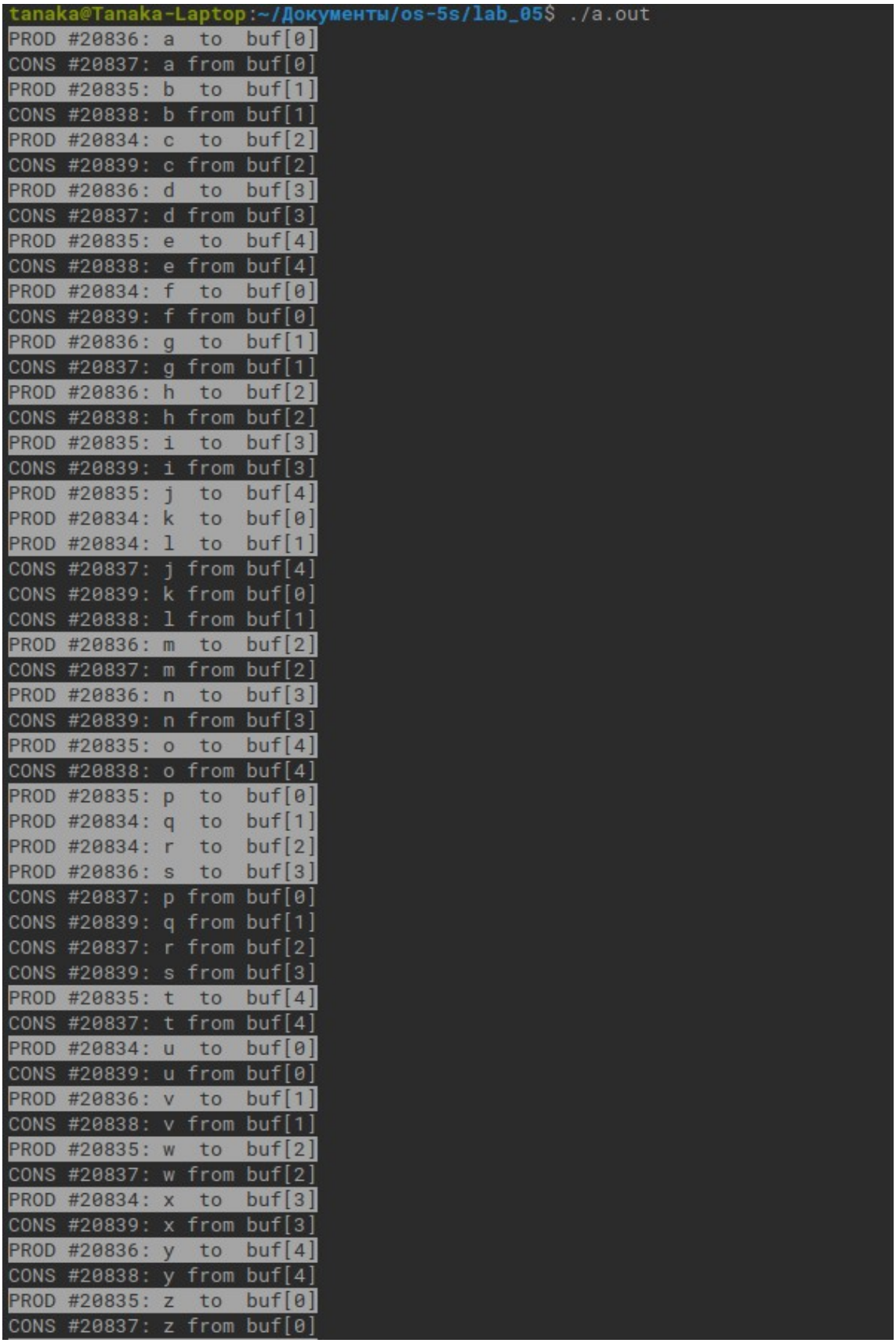
```
// Производство-потребление
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#define CNT_BUF 5
#define VALUES 10

#define PROD 3
#define CONS 3

#define BIN 0
#define EMPTY 1
#define FULL 2

struct sembuf producer_P[2] = {{EMPTY, -1, 0}, {FULL, -1, 0}};
struct sembuf producer_V[2] = {{BIN, 1, 0}, {FULL, 1, 0}};
struct sembuf consumer_P[2] = {{BIN, -1, 0}, {FULL, -1, 0}};
struct sembuf consumer_V[2] = {{EMPTY, 1, 0}, {FULL, 1, 0}};
```

```
int *shared_buffer;
int *shared_pos_consumer;
int *shared_pos_producer;
int *shared_letter;

int producer(int semaphores, pid_t pid)
{
   while (1)
   {
      sleep(rand() % 5);
      if (semop(semaphores, producer_P, 2) == -1)
         {
            perror("Can't SEMOP!\n");
            exit(1);
         }

      shared_buffer[*shared_pos_producer] = *shared_letter + 'a';
      printf("\x1b[7mPROD #%d: %c  to  buf[%d]\n", pid, shared_buffer[*shared_pos_producer],
*shared_pos_producer);
      (*shared_pos_producer) = (*shared_pos_producer + 1) % CNT_BUF;

      (*shared_letter) = (*shared_letter + 1) % 26;

      if (semop(semaphores, producer_V, 2) == -1)
         {
            perror("Can't SEMOP!\n");
            exit(1);
         }
   }
   return 0;
}

int consumer(int semaphores, pid_t pid)
{
   while (1)
   {
      sleep(rand() % 2);
      if (semop(semaphores, consumer_P, 2) == -1)
         {
            perror("Can't SEMOP!\n");
            exit(1);
         }

      printf("\x1b[0mCONS #%d: %c from buf[%d]\n", pid, shared_buffer[*shared_pos_consumer],
*shared_pos_consumer);
      (*shared_pos_consumer) = (*shared_pos_consumer + 1) % CNT_BUF;

      if (semop(semaphores, consumer_V, 2) == -1)
         {
            perror("Can't SEMOP!\n");
            exit(1);
         }
   }
   return 0;
```

```
}

int main()
{
   srand(NULL);
   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("Can't SHMGET!\n");
      exit(1);
   }

   shared_pos_producer = shmat(fd, 0, 0);
   if (*shared_pos_producer == -1)
   {
      perror("Can't SHMAT!\n");
      exit(1);
   }

   shared_buffer = shared_pos_producer + 3 * sizeof(char);
   shared_pos_consumer = shared_pos_producer + 2 * sizeof(char);
   shared_letter = shared_pos_producer + sizeof(char);
   (*shared_pos_producer) = 0;
   (*shared_pos_consumer) = 0;
   (*shared_letter) = 0;

   int semaphores = semget(IPC_PRIVATE, 3, IPC_CREAT | perms);
   if (semaphores == -1)
      {
         perror("Can't SEMGET!\n");
         exit(1);
      }

   int ctlb = semctl(semaphores, BIN, SETVAL, 0);
   int ctle = semctl(semaphores, EMPTY, SETVAL, CNT_BUF);
   int ctlf = semctl(semaphores, FULL, SETVAL, 1);

   if (ctlf == -1 || ctle == -1 || ctlb == -1)
   {
      perror("Can't SEMCTL!\n");
      exit(1);
   }

   pid_t pid;
   for (int i = 0; i < PROD && pid != 0; i++)
   {
      if ((pid = fork()) == -1)
      {
         perror("Can't FORK prod!\n");
         exit(1);
      }
      if (pid == 0)
```

```
        {
            producer(semaphores, getpid());
            return 0;
        }
    }

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

        if ((pid = fork()) == -1)
        {
            perror("Can't FORK cons!\n");
            exit(1);
        }
        if (pid == 0)
        {
            consumer(semaphores, getpid());
            return 0;
        }
    }

    if (pid != 0)
    {
        int status;
        for (int i = 0; i < CNT_BUF; i++)
            wait(&status);

        if (shmdt(shared_pos_producer) == -1)
        {
            perror("Can't SHMDT!\n");
            exit(1);
        }
    }
    return 0;
}
```

Пример работы программы:



Изображение 1. – Работа программы.

Задание 2

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

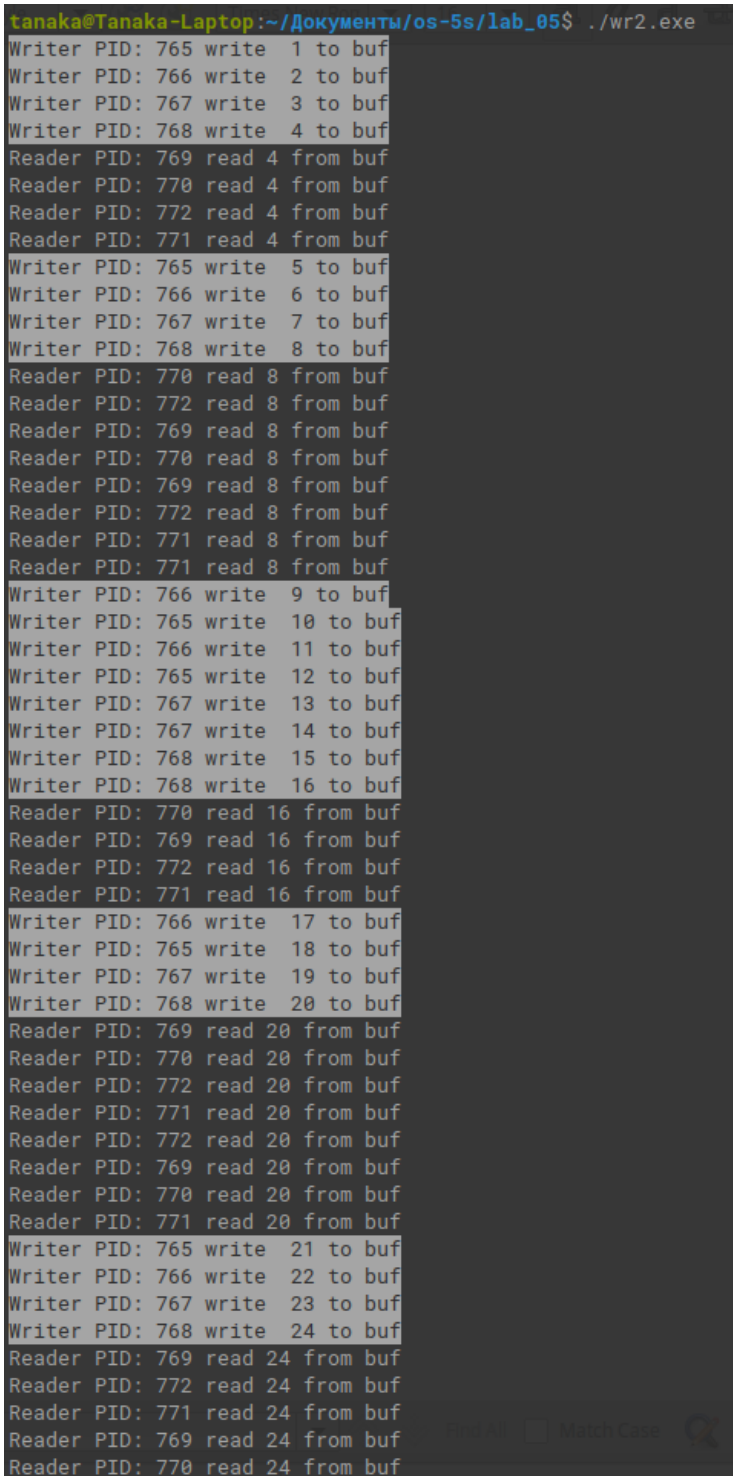
Код программы:

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

// Количество читателей и писателей
#define COUNT_READERS 4
#define COUNT_WRITERS 4

// Операции на семафорах
#define SEM_OP_INC 1
#define SEM_OP_DEC -1
#define SEM_OP_WAIT 0

// Ключи
#define ACTIVE_READERS 0
#define ACTIVE_WRITER 1
#define WAITING_WRITERS 2
#define WAITING_READERS 3


struct sembuf start_writer[] = {{WAITING_WRITERS,   SEM_OP_INC, 0},
                                {ACTIVE_READERS,    SEM_OP_WAIT, 0},
                                {ACTIVE_WRITER,     SEM_OP_WAIT, 0},
                                {ACTIVE_WRITER,     SEM_OP_INC, 0},
                                {WAITING_WRITERS,   SEM_OP_DEC, 0},},
               stop_writer[] = {{ACTIVE_WRITER, -1, 0}},
              start_reader[] = {{WAITING_READERS,   SEM_OP_INC, 0},
                                {WAITING_WRITERS,   SEM_OP_WAIT, 0},
                                {ACTIVE_WRITER,     SEM_OP_WAIT, 0},
                                {ACTIVE_READERS,    SEM_OP_INC, 0},
                                {WAITING_READERS,   SEM_OP_DEC, 0},},
               stop_reader[] = {{ACTIVE_READERS, -1, 0}};


void Reader(int nomer, int sem_id, int* buf)
{
    while (1)
    {
        semop(sem_id, start_reader, 5);

        printf("Reader #%d -> ", nomer);
        printf("read %d\n", *buf);

        semop(sem_id, stop_reader, 1);

        sleep(rand() % 2);
    }
```

```
}

void Writer(int nomer, int sem_id, int* buf)
{
    while (1)
    {
        semop(sem_id, start_writer, 5);

        printf("Writer #%d -> ", nomer);
        printf("write %d\n", ++(*buf));

        semop(sem_id, stop_writer, 1);

        sleep(rand() % 3);
    }
}

int main()
{
    int perms = S_IRWXU | S_IRWXG | S_IRWXO;
    int shm_id;
    int sem_id;
    int *mem_ptr;

    // возвращает идентификатор разделяемому сегменту памяти
    if ((shm_id = shmget(IPC_PRIVATE, sizeof(int), IPC_CREAT | perms)) == -1)
    {
        perror("Unable to create a shared area.\n");
        return 1;
    }

    // возвращает указатель на сегмент разделяемой памяти
    if ((mem_ptr = shmat(shm_id, NULL, 0)) == -1)
    {
        perror("Can't attach memory.\n");
        return 1;
    }

    // создание набора семафоров
    if ((sem_id = semget(IPC_PRIVATE, 4, perms)) == -1)
    {
        perror("Can’t semget.\n");
        return 1;
    }

    // изменение управляющих параметров набора семафоров
    semctl(sem_id, ACTIVE_READERS,  SETVAL, 0);
    semctl(sem_id, ACTIVE_WRITER,   SETVAL, 0);
    semctl(sem_id, WAITING_READERS, SETVAL, 0);
    semctl(sem_id, WAITING_WRITERS, SETVAL, 0);

    // создание процессов
    int count_processes = 0;
    pid_t pid;

    *mem_ptr = 0;

    for (int i = 0; i < COUNT_WRITERS; i++)
    {
        if ((pid = fork())== -1)
        {
            perror("Can’t fork.\n");
            return 1;
        }

        if (!pid)
            Writer(i + 1, sem_id, mem_ptr);
        else
```

```
            count_processes++;
        }

        for (int i = 0; i < COUNT_READERS; i++)
        {
            if ((pid = fork())== -1)
            {
                perror("Can’t fork.\n");
                return 1;
            }

            if (!pid)
                Reader(i + 1, sem_id, mem_ptr);
            else
                count_processes++;
        }

        // ожидание завершение процессов
        int status;
        for (int i = 0; i < count_processes; i++)
        {
            wait(&status);
            if (!WIFEXITED(status))
                printf("exit-error, code = %d\n", status);
        }

        // освобождение ресурсов
        shmdt(mem_ptr);
        semctl(sem_id, 0, IPC_RMID);
        shmctl(shm_id, IPC_RMID, 0);

        return 0;
    }
```

Пример работы программы:



Изображение 3. – Работа второй программы.