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» (ИУ7)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.04 Программная инженерия**

О Т Ч Е Т

по лабораторной работе № 5

Название: Взаимодействие параллельных процессов.

Дисциплина: Операционные системы

Студент ИУ7-53Б _____ А.С.Саркисов
(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель _____ Н.Ю. Рязанова
(Подпись, дата) (И.О. Фамилия)

Москва, 2020

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.

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>

const int totalProducers = 5;
const int totalConsumers = 5;
const int bufferSize = 10;
const int perm = S_IRWXU | S_IRWXG | S_IRWXO;

char* sharedBuffer;
char* posToWrite;
```

```
char* posToRead;

#define SB 0
#define SE 1
#define SF 2

#define P -1
#define V 1

struct sembuf producerStart[2] = { {SE, P, 0}, {SB, P, 0} };
struct sembuf producerStop[2] =  { {SB, V, 0}, {SF, V, 0} };
struct sembuf consumerStart[2] = { {SF, P, 0}, {SB, P, 0} };
struct sembuf consumerStop[2] =  { {SB, V, 0}, {SE, V, 0} };



void producer(const int semid, const int consumerNum)
{
   for (int i = 0; i < 26; i++) {
      char symb = i + 65;

      sleep(rand() % 4);

      int semP = semop(semid, producerStart, 2);
      if (semP == -1)
            {
         perror("Can't make operation on semaphors.");
         exit(1);
      }

      sharedBuffer[(*posToWrite) % bufferSize] = symb;
```

```
            printf("Producer #%d ----> %c\n", consumerNum, sharedBuffer[(*posToWrite) %
bufferSize]);
            (*posToWrite)++;


        int semV = semop(semid, producerStop, 2);
            if (semV == -1)
            {
                perror("Can't make operation on semaphors.");
                exit(1);
            }
    }
}

void consumer(const int semid, const int consumerNum)
{
    for (int i = 0; i < 26; i++) {
        char symb = i + 65;


        sleep(rand() % 2);


        int semP = semop(semid, consumerStart, 2);
            if (semP == -1)
            {
                perror("Can't make operation on semaphors.");
                exit(1);
            }


            printf("Consumer #%d <---- %c\n", consumerNum, sharedBuffer[(*posToRead) %
bufferSize]);
            (*posToRead)++;
```

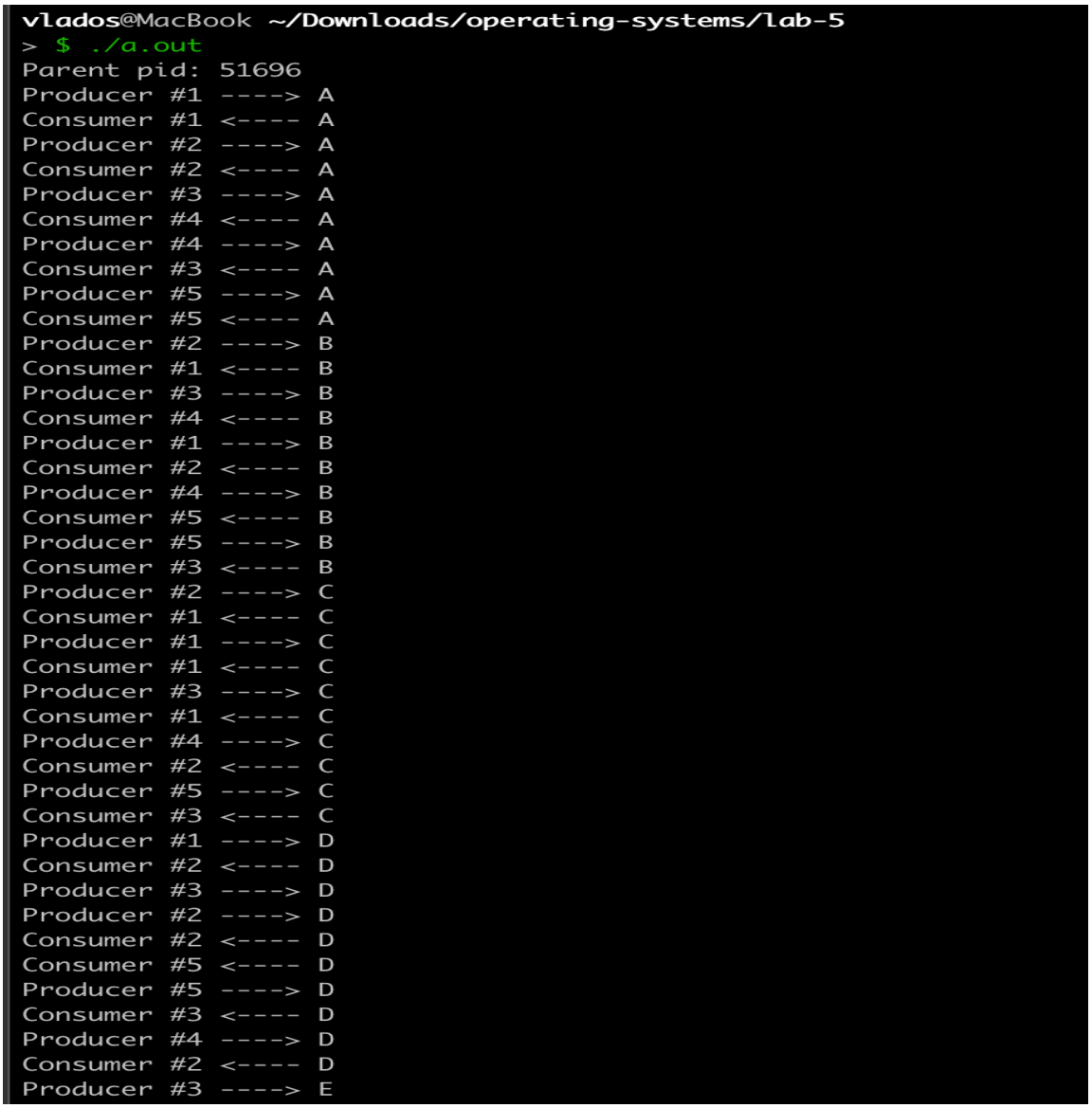
```
        int semV = semop(semid, consumerStop, 2);
        if (semV == -1)
        {
            perror("Can't make operation on semaphors.");
            exit(1);
        }
    }
}

int main()
{
        int shmid, semid;

        int parent_pid = getpid();
        printf("Parent pid: %d\n", parent_pid);

        if ((shmid = shmget(IPC_PRIVATE, (bufferSize + 2) * sizeof(char), IPC_CREAT |
perm)) == -1)
        {
                perror("Unable to create a shared area.");
                exit(1);
        }

        sharedBuffer = shmat(shmid, 0, 0);
        if (*sharedBuffer == -1)
        {
                perror("Can't attach memory");
                exit(1);
        }
```

```
posToWrite = sharedBuffer;
        posToRead = sharedBuffer + sizeof(char);
sharedBuffer = sharedBuffer + sizeof(char) * 2;


        *posToWrite = 0;
*posToRead = 0;


        if ((semid = semget(IPC_PRIVATE, 3, IPC_CREAT | perm)) == -1)
        {
                perror("Unable to create a semaphore.");
                exit( 1 );
        }

        int ret_sb = semctl(semid, SB, SETVAL, 1);
        int ret_se = semctl(semid, SE, SETVAL, bufferSize);
        int ret_sf = semctl(semid, SF, SETVAL, 0);

        if ( ret_se == -1 || ret_sf == -1 || ret_sb == -1)
        {
                perror("Can't set control semaphors.");
                exit(1);
        }

        pid_t pid;
        for (size_t i = 0; i < totalProducers; i++)
        {
                switch (pid = fork())
                {
                case -1:
                        printf("Can't fork.");
```

```
                exit(1);
            case 0:
                producer(semid, i + 1);
                exit(0);
            }
        }


    for (size_t i = 0; i < totalConsumers; i++)
        {
            switch (pid = fork())
            {
            case -1:
                printf("Can't fork.");
                exit(1);
            case 0:
                consumer(semid, i + 1);
                exit(0);
            }
        }

        for (size_t i = 0; i < totalConsumers + totalProducers; i++)
        {
            int status;
            pid_t ret_value = wait(&status);

            printf("Child process has finished: PID = %d, status = %d\n", ret_value,
status);

            if (WIFEXITED(status))
```

```
                printf("Child %d finished with %d code.\n\n", ret_value,
WEXITSTATUS(status));
            else if (WIFSIGNALED(status))
                printf("Child %d finished from signal with %d code.\n\n", ret_value,
WTERMSIG(status));
            else if (WIFSTOPPED(status))
                printf("Child %d finished from signal with %d code.\n\n", ret_value,
WSTOPSIG(status));
        }


    printf("%d\n", *posToWrite);
    if (shmdt(posToWrite) == -1)
        {
            perror("Can't detach shared memory");
            exit(1);
        }


    exit(0);
}
```

Работа 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.

Писателей д.б. не меньше 3х, а читателей не меньше 5ти. Для реализации взаимоисключения используются семафоры.

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 TOTAL_WRITERS 5
#define TOTAL_READERS 5

#define ACTIVE_READER 0
#define ACTIVE_WRITER 1
#define BIN_ACTIVE_WRITER 2
#define WAITING_WRITER 3

struct sembuf startRead[] = { { WAITING_WRITER, 0, 0 },
                              { ACTIVE_WRITER,  0, 0 },
                              { ACTIVE_READER,  1, 0 } };
```

```
struct sembuf  stopRead[] = { {ACTIVE_READER, -1, 0} };

struct sembuf  startWrite[] = { { WAITING_WRITER,     1, 0 },
                                { ACTIVE_READER,      0, 0 },
                                { BIN_ACTIVE_WRITER, -1, 0 },
                                { ACTIVE_WRITER,      1, 0 },
                                { WAITING_WRITER,    -1, 0 } };

struct sembuf  stopWrite[] = { { ACTIVE_WRITER,    -1, 0 },
                               { BIN_ACTIVE_WRITER, 1, 0 }};

const int perm =  S_IRWXU | S_IRWXG | S_IRWXO;

void writer(int semid, int* buffer, int num) {
   while (1)
   {
      semop(semid, startWrite, 5);
      (*buffer)++;
                  printf("Writer #%d ----> %d\n", num, *buffer);
      semop(semid, stopWrite, 2);
                  sleep(rand() % 4);
   }
}

void reader(int semid, int* buffer, int num) {
   while (1)
   {
```

```
        semop(semid, startRead, 3);
                printf("Reader #%d <---- %d\n", num, *buffer);
        semop(semid, stopRead, 1);
                sleep(rand() % 2);
    }
}

int main()
{
    srand(time(NULL));

    int parent_pid = getpid();
        printf("Parent pid: %d\n", parent_pid);

    int shm_id;
    if ((shm_id = shmget(IPC_PRIVATE, sizeof(int), IPC_CREAT | perm)) == -1)
        {
                perror("Unable to create a shared area.");
                exit(1);
        }

    int *sharedBuffer = shmat(shm_id, 0, 0);
    if (sharedBuffer == (void*) -1)
    {
        perror("Can't attach memory");
        exit(1);
    }
```

```
(*sharedBuffer) = 0;

int sem_id;
if ((sem_id = semget(IPC_PRIVATE, 4, IPC_CREAT | perm)) == -1)
        {
                perror("Unable to create a semaphore.");
                exit(1);
        }

        int retValue = semctl(sem_id, BIN_ACTIVE_WRITER, SETVAL, 1);
if (retValue == -1)
        {
                perror( "Can't set control semaphors.");
                exit(1);
        }

        pid_t pid = -1;

        for (int i = 0; i < TOTAL_WRITERS && pid != 0; i++) {
    pid = fork();
    if (pid == -1) {
        perror("Writer's fork error.");
        exit(1);
    }
    if (pid == 0) {
        writer(sem_id, sharedBuffer, i);
    }
        }
```

```
for (int i = 0; i < TOTAL_READERS && pid != 0; i++) {
    pid = fork();
    if (pid == -1) {
        perror("Reader's fork error.");
        exit(1);
    }
    if (pid == 0) {
        reader(sem_id, sharedBuffer, i);
    }
        }

if (shmdt(sharedBuffer) == -1) {
    perror("Can't detach shared memory");
    exit(1);
}

if (pid != 0) {
    int *status;

    for (int i = 0; i < TOTAL_READERS + TOTAL_WRITERS; ++i) {
            wait(status);
    }

    if (shmctl(shm_id, IPC_RMID, NULL) == -1) {
        perror("Can't free memory!");
        exit(1);
    }
```

```
    }


    exit(0);
}
```

Работа программы

```
vlados@MacBook ~/Downloads/operating-systems/lab-5
> $ ./a.out
Parent pid: 51977
Writer #0 ----> 1
Writer #1 ----> 2
Writer #2 ----> 3
Writer #3 ----> 4
Writer #4 ----> 5
Reader #0 <---- 5
Reader #1 <---- 5
Reader #2 <---- 5
Reader #3 <---- 5
Reader #4 <---- 5
Reader #0 <---- 5
Reader #1 <---- 5
Reader #3 <---- 5
Reader #4 <---- 5
Reader #2 <---- 5
Reader #0 <---- 5
Reader #1 <---- 5
Reader #4 <---- 5
Reader #3 <---- 5
Reader #2 <---- 5
Reader #1 <---- 5
Reader #4 <---- 5
Reader #0 <---- 5
Reader #3 <---- 5
Reader #2 <---- 5
Writer #0 ----> 6
Writer #1 ----> 7
Writer #4 ----> 8
Writer #2 ----> 9
Writer #3 ----> 10
Reader #1 <---- 10
Reader #3 <---- 10
Reader #4 <---- 10
Reader #0 <---- 10
Reader #2 <---- 10
Reader #1 <---- 10
Reader #0 <---- 10
```