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
по лабораторной работе № 5

Дисциплина: Операционные системы

Тема: Взаимодействие параллельных процессов

Студент: Платонова Ольга

Группа: ИУ7-55Б

Преподаватели: Рязанова Н. Ю.

Москва, 2020 г.

Оглавление

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.

1. Файл errors.h

```
#ifndef EXIT_CODES_H
#define EXIT_CODES_H
enum ExitCodes
{
    SUCCESS,

    SHMGET_ERR,
    SHMAT_ERR,
    SHMCTL_ERR,

    SEMGET_ERR,
    SEMOP_ERR,
    SEMCTL_ERR,

    FORK_ERR
};
#endif
```

2. Файл prod_cons.c

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

#include "errors.h"

const int BUFSIZE = 24;
static char* shmbuf = NULL;
const char ALPHABET[] = "abcdefghijklmnopqrstuvwxyz";
const int AlphLen = strlen(ALPHABET);

static char* idxProd = 0;
static char* idxCons = 0;

static bool wasSignal = false;
static bool isEndConsume = false;

enum Semaphores
{
	BUFF_FULL,
	BUFF_EMPTY,
	BIN_SEM
};

struct sembuf prodP[2] = {{BUFF_EMPTY, -1, 0}, {BIN_SEM, -1, 0}};
struct sembuf prodV[2] = {{BIN_SEM, 1, 0}, {BUFF_FULL, 1, 0}};

struct sembuf consP[2] = {{BUFF_FULL, -1, 0}, {BIN_SEM, -1, 0}};
struct sembuf consV[2] = {{BIN_SEM, 1, 0}, {BUFF_EMPTY, 1, 0}};
```

```
const int perms = S_IRWXU | S_IRWXG | S_IRWXO;

const int NumProd = 3;
const int NumCons = 3;

void producer(const int semfd, const int id);
void consumer(const int semfd, const int id);

void errorCheck(int fd, int err, char* msg, int exCode);
void catchSignal(int signalNum);

int main()
{
	//память//
	int size = (BUFSIZE + 2) * sizeof(char);
	int shmfd = shmget(IPC_PRIVATE, size, IPC_CREAT | perms);
	errorCheck(shmfd, -1, "shmget", SHMGET_ERR);

	idxProd = (char*) shmat(shmfd, 0, 0);
	if (idxProd == (char*) -1) {
		perror("shmat");
		return SHMAT_ERR;
	}

	idxCons = idxProd + sizeof(char);
	shmbuf = idxCons + sizeof(char);

	//семафоры //
	int semfd = semget(IPC_PRIVATE, 3, IPC_CREAT | perms);
	errorCheck(semfd, -1, "semget", SEMGET_ERR);

	int s1 = semctl(semfd, BUFF_FULL, SETVAL, 0);
	errorCheck(s1, -1, "semctl", SEMCTL_ERR);

	int s2 = semctl(semfd, BUFF_EMPTY, SETVAL, BUFSIZE);
	errorCheck(s2, -1, "semctl", SEMCTL_ERR);

	int s3 = semctl(semfd, BIN_SEM, SETVAL, 1);
	errorCheck(s3, -1, "semctl", SEMCTL_ERR);
```

```
    signal(SIGINT, catchSignal);

    //процессы//
    //производители
    for (int i = 0; i < NumProd; i++) {
        pid_t pid = fork();
        errorCheck(pid, -1, "fork", FORK_ERR);

        if (pid == 0) {
            producer(semfd, i);
            exit(0);
        }
    }

    //потребители
    for (int i = 0; i < NumCons; i++) {
        pid_t pid = fork();
        errorCheck(pid, -1, "fork", FORK_ERR);

        if (pid == 0) {
            consumer(semfd, i);
            exit(0);
        }
    }

    //завершение//
    for (int i = 0; i < NumProd + NumCons; i++) {
        int status;
        wait(&status);
    }

    s1 = shmctl(shmfd, IPC_RMID, NULL);
    errorCheck(s1, -1, "shmctl", SHMCTL_ERR);

    s2 = semctl(semfd, BIN_SEM, IPC_RMID, 0);
    errorCheck(s2, -1, "semctl", SEMCTL_ERR);

    return 0;
}
```

```
void producer(const int semfd, const int id)
{
	while(!wasSignal) {
		int op1 = semop(semfd, prodP, 2);
		errorCheck(op1, -1, "Producer: semop1", SEMOP_ERR);

		shmbuf[*idxProd] = ALPHABET[*idxProd];
		printf("Producer %d:\t (pid %d)\t %c --> \n", id + 1, getpid(), ALPHABET[*idxProd]);
		(*idxProd)++;

		int op2 = semop(semfd, prodV, 2);
		errorCheck(op2, -1, "Producer: semop2", SEMOP_ERR);

		sleep(rand() % 3);
	}
}

void consumer(const int semfd, const int id)
{
	while(!isEndConsume) {
		int op1 = semop(semfd, consP, 2);
		errorCheck(op1, -1, "Consumer: semop1", SEMOP_ERR);

		printf("Consumer %d:\t (pid %d)\t <-- %c\n", id + 1, getpid(), shmbuf[*idxCons]);
		(*idxCons)++;

		int op2 = semop(semfd, consV, 2);
		errorCheck(op2, -1, "Consumer: semop2", SEMOP_ERR);

		sleep(rand() % 5);

		isEndConsume = (*idxCons == *idxProd);
	}
}

void errorCheck(int fd, int err, char* msg, int exCode)
{
	if (fd == err) {
		perror(msg);
```

```
            exit(exCode);
        }
}

void catchSignal(int signalNum)
{
        printf("");
        wasSignal = true;
}
```

Результат работы программы.

```
Producer 1:      (pid 20120)     a -->
Producer 2:      (pid 20121)     b -->
Producer 3:      (pid 20122)     c -->
Consumer 1:      (pid 20123)     <-- a
Consumer 2:      (pid 20124)     <-- b
Consumer 3:      (pid 20125)     <-- c
Producer 1:      (pid 20120)     d -->
Producer 2:      (pid 20121)     e -->
Producer 3:      (pid 20122)     f -->
Producer 1:      (pid 20120)     g -->
Producer 1:      (pid 20120)     h -->
Producer 2:      (pid 20121)     i -->
Producer 2:      (pid 20121)     j -->
Producer 3:      (pid 20122)     k -->
Producer 3:      (pid 20122)     l -->
^CConsumer 2:    (pid 20124)     <-- d
Consumer 1:      (pid 20123)     <-- e
Consumer 3:      (pid 20125)     <-- f
Consumer 2:      (pid 20124)     <-- g
Consumer 1:      (pid 20123)     <-- h
Consumer 3:      (pid 20125)     <-- i
Consumer 2:      (pid 20124)     <-- j
Consumer 1:      (pid 20123)     <-- k
Consumer 2:      (pid 20124)     <-- l
```

В приведенном примере сигнал поступил после записи буквы l.

Отметим, что при поступлении сигнала CTRL+C, производители прекращают запись, потребители завершают чтение произведенных элементов.

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.

Для реализации взаимоисключения используются семафоры.

Код программы.

1. Файл errors.h

```
#ifndef EXIT_CODES_H
#define EXIT_CODES_H
enum ExitCodes
{
    SUCCESS,

    SHMGET_ERR,
    SHMAT_ERR,
    SHMCTL_ERR,
    SHMDT_ERR,

    SEMGET_ERR,
    SEMOP_ERR,
    SEMCTL_ERR,

    FORK_ERR
};
#endif
```

2. Файл read_write.c

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

#include "errors.h"

static int* shmbuf = NULL;

enum Semaphores
{
	ACTIVE_WR,
	WAITING_WR,

	ACTIVE_RD,
	WAITING_RD
};

struct sembuf startWrite[5] = {{WAITING_WR, 1, 0},
                               {ACTIVE_RD, 0, 0},
                               {ACTIVE_WR, 0, 0},
                               {ACTIVE_WR, 1, 0},
                               {WAITING_WR, -1, 0}};
struct sembuf stopWrite[1] = {{ACTIVE_WR, -1, 0}};

struct sembuf startRead[5] = {{WAITING_RD, 1, 0},
                              {ACTIVE_WR, 0, 0},
                              {WAITING_WR, 0, 0},
                              {ACTIVE_RD, 1, 0},
                              {WAITING_RD, -1, 0}};
```

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

const int perms = S_IRWXU | S_IRWXG | S_IRWXO;

const int NumReaders = 5;
const int NumWriters = 3;
const int LastRecord = 10;

static bool isEndRead = false;
static bool isEndWrite = false;

void writer(const int semfd, const int id);
void reader(const int semfd, const int id);
void errorCheck(int fd, int err, char* msg, int exCode);

int main()
{
	//память//
	int size = sizeof(int);
	int shmfd = shmget(IPC_PRIVATE, size, IPC_CREAT | perms);
	errorCheck(shmfd, -1, "shmget", SHMGET_ERR);

	shmbuf = (int*) shmat(shmfd, 0, 0);
	if (shmbuf == (int*) -1) {
		perror("shmat");
		return SHMAT_ERR;
	}

	(*shmbuf) = 0;

	//семафоры//
	int semfd = semget(IPC_PRIVATE, 4, IPC_CREAT | perms);
	errorCheck(semfd, -1, "semget", SEMGET_ERR);

	int s1 = semctl(semfd, ACTIVE_WR, SETVAL, 0);
	errorCheck(s1, -1, "semctl", SEMCTL_ERR);

	int s2 = semctl(semfd, WAITING_WR, SETVAL, 0);
	errorCheck(s2, -1, "semctl", SEMCTL_ERR);
```

```
    int s3 = semctl(semfd, ACTIVE_RD, SETVAL, 0);
    errorCheck(s3, -1, "semctl", SEMCTL_ERR);

    int s4 = semctl(semfd, WAITING_RD, SETVAL, 0);
    errorCheck(s4, -1, "semctl", SEMCTL_ERR);

    //процессы//
    //писатели
    for (int i = 0; i < NumWriters; i++) {
        pid_t pid = fork();
        errorCheck(pid, -1, "fork", FORK_ERR);

        if (pid == 0) {
            writer(semfd, i);
            exit(0);
        }
    }

    //читатели
    for (int i = 0; i < NumReaders; i++) {
        pid_t pid = fork();
        errorCheck(pid, -1, "fork", FORK_ERR);

        if (pid == 0) {
            reader(semfd, i);
            exit(0);
        }
    }

    //завершение//
    for (int i = 0; i < NumWriters + NumReaders; i++) {
        int status;
        wait(&status);
    }

    int s = shmdt(shmbuf);
    errorCheck(s, -1, "shmdt", SHMDT_ERR);
    return 0;
}
```

```
void writer(const int semfd, const int id)
{
	while(!isEndWrite) {
		int op1 = semop(semfd, startWrite, 5);
		errorCheck(op1, -1, "Writer: semop1", SEMOP_ERR);

		if ((*shmbuf) < LastRecord) {
			printf("Writer %d:\t (pid %d)\t %d --> \n", id + 1, getpid(), ++(*shmbuf));
		}
		isEndWrite = ((*shmbuf) >= LastRecord);

		int op2 = semop(semfd, stopWrite, 1);
		errorCheck(op2, -1, "Producer: semop3", SEMOP_ERR);

		sleep((rand() % 3) + 1);
	}
}

void reader(const int semfd, const int id)
{
	while(!isEndRead) {
		int op1 = semop(semfd, startRead, 5);
		errorCheck(op1, -1, "Reader: semop1", SEMOP_ERR);

		printf("Reader %d:\t (pid %d)\t <-- %d\n", id + 1, getpid(), *shmbuf);
		isEndRead = ((*shmbuf) >= LastRecord);

		int op2 = semop(semfd, stopRead, 1);
		errorCheck(op2, -1, "Reader: semop2", SEMOP_ERR);

		sleep((rand() % 6) + 1);
	}
}

void errorCheck(int fd, int err, char* msg, int exCode)
{
	if (fd == err) {
		perror(msg);
		exit(exCode);
```

```
    }
}
```

Результат работы программы.

```
Writer 1:       (pid 5904)      1 -->
Writer 2:       (pid 5905)      2 -->
Writer 3:       (pid 5906)      3 -->
Reader 2:       (pid 5908)      <-- 3
Reader 4:       (pid 5910)      <-- 3
Reader 1:       (pid 5907)      <-- 3
Reader 3:       (pid 5909)      <-- 3
Reader 5:       (pid 5911)      <-- 3
Writer 1:       (pid 5904)      4 -->
Writer 2:       (pid 5905)      5 -->
Writer 3:       (pid 5906)      6 -->
Reader 4:       (pid 5910)      <-- 6
Reader 2:       (pid 5908)      <-- 6
Reader 1:       (pid 5907)      <-- 6
Reader 3:       (pid 5909)      <-- 6
Reader 5:       (pid 5911)      <-- 6
Writer 1:       (pid 5904)      7 -->
Writer 2:       (pid 5905)      8 -->
Writer 3:       (pid 5906)      9 -->
Writer 1:       (pid 5904)      10 -->
Reader 2:       (pid 5908)      <-- 10
Reader 4:       (pid 5910)      <-- 10
Reader 1:       (pid 5907)      <-- 10
Reader 3:       (pid 5909)      <-- 10
Reader 5:       (pid 5911)      <-- 10
```