## 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)

**ОТЧЕТ**

**по лабораторной работе №** 5

**Взаимодействие параллельных потоков**

**Дисциплина: Операционные системы**

|  |  |  |  |
| --- | --- | --- | --- |
| Студент | ИУ7-56Б  (Группа) | (Подпись, дата) | Ж. Р.Турсунов  (И.О. Фамилия) |
| Преподаватель |  | (Подпись, дата) | Н.Ю.Рязанова  (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.

# Листинг кода:

#include <stdio.h>

#include <stdlib.h> // rand

#include <unistd.h> // sleep

#include <signal.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <sys/ipc.h>

#include <sys/sem.h>

#include <sys/shm.h>

#include <sys/stat.h>

const int N **=** 9**;**

// const int COUNT = 5;

const size\_t shm\_size **=** **(**N **+** 2**)** **\*** **sizeof(**int**);**

int**\*** shm\_attached\_address**;**

int**\*** shm\_buffer**;**

int**\*** shm\_pos\_consumer**;**

int**\*** shm\_pos\_producer**;**

#define SEM\_BINARY 0

#define SEM\_EMPTY 1

#define SEM\_FULL 2

#define P -1

#define V 1

struct sembuf producer\_start**[**2**]** **=** **{** **{** SEM\_EMPTY **,** P**,** 0 **},** **{** SEM\_BINARY**,** P**,** 0 **}** **};**

struct sembuf producer\_stop **[**2**]** **=** **{** **{** SEM\_BINARY**,** V**,** 0 **},** **{** SEM\_FULL **,** V**,** 0 **}** **};**

struct sembuf consumer\_start**[**2**]** **=** **{** **{** SEM\_FULL **,** P**,** 0 **},** **{** SEM\_BINARY**,** P**,** 0 **}** **};**

struct sembuf consumer\_stop **[**2**]** **=** **{** **{** SEM\_BINARY**,** V**,** 0 **},** **{** SEM\_EMPTY **,** V**,** 0 **}** **};**

void producer**(**const int semid**,** const int value**,** const int producer\_id**)** **{**

sleep**(**rand**()** **%** 3**);**

**if** **(**semop**(**semid**,** producer\_start**,** 2**)** **==** **-**1**)** **{**

puts**(**"PRODUCER can not make operation on semaphore"**);**

exit**(**EXIT\_FAILURE**);**

**}**

char cur = value;

shm\_buffer**[\***shm\_pos\_producer**]** **=** cur**;**

printf**(**"PRODUCER %d pos %d -----> produced %d\n"**,** producer\_id**,** **\***shm\_pos\_producer**,** shm\_buffer**[\***shm\_pos\_producer**]);**

**(\***shm\_pos\_producer**)++;**

**if** **(**semop**(**semid**,** producer\_stop**,** 2**)** **==** **-**1**)** **{**

puts**(**"PRODUCER can not make operation on semaphore"**);**

exit**(**EXIT\_FAILURE**);**

**}**

**}**

void consumer**(**const int semid**,** const int value**,** const int consumer\_id**)** **{**

sleep**(**rand**()** **%** 2**);**

**if** **(**semop**(**semid**,** consumer\_start**,** 2**)** **==** **-**1**)** **{**

puts("PRODUCER can not make operation on semaphore");

exit(EXIT\_FAILURE);

}

printf("CONSUMER %d pos %d <----- cunsumed %c\n", consumer\_id, \*shm\_pos\_consumer, shm\_buffer[\*shm\_pos\_consumer]);

(\*shm\_pos\_consumer)++;

if (semop(semid, consumer\_stop, 2) == -1) {

puts("PRODUCER can not make operation on semaphore");

exit(EXIT\_FAILURE);

}

}

void make\_producer(const int id, const int semid, int value, int COUNT) {

pid\_t child1\_pid;

if ((child1\_pid = fork()) == -1) {

puts("Can't fork");

exit(EXIT\_FAILURE);

}

if(child1\_pid == 0) {

// child

printf("Created producer %d\n", id);

for (int i = 0; i < COUNT; i++) {

producer(semid, value, id);

value++;

}

printf("PRODUCER %d finished his work. Terminating..\n", id);

exit(EXIT\_SUCCESS);

}

}

void make\_consumer(const int id, const int semid, int COUNT) {

pid\_t child\_pid;

if ((child\_pid = fork()) == -1) {

puts("Can't fork");

exit(EXIT\_FAILURE);

}

if(child\_pid == 0) {

printf("Created consumer %d\n", id);

// child

for (int i = 0; i < COUNT; i++) {

consumer(semid, i, id);

}

printf("CONSUMER %d finished his work. Terminating..\n", id);

exit(EXIT\_SUCCESS);

}

}

// 3 семафора

int main() {

srand(0);

int shmid; // shared memory id

int semid; // semaphore id

pid\_t parent\_pid = getpid();

printf("Parent pid: %i\n", parent\_pid);

if ((shmid = shmget(IPC\_PRIVATE, shm\_size, IPC\_CREAT | S\_IRWXU | S\_IRWXG | S\_IRWXO)) == -1) {

puts("Unable to create shared area");

exit(EXIT\_FAILURE);

}

shm\_attached\_address = shmat(shmid, NULL, 0);

if (\*(char\*)shm\_attached\_address == -1) {

puts("Can not attach memory");

exit(EXIT\_FAILURE);

}

shm\_pos\_producer = shm\_attached\_address;

shm\_pos\_consumer = shm\_attached\_address + sizeof(int);

shm\_buffer = shm\_attached\_address + 2 \* sizeof(int);

(\*shm\_pos\_producer) = 0;

(\*shm\_pos\_consumer) = 0;

if ((semid = semget(IPC\_PRIVATE, 3, IPC\_CREAT | S\_IRWXU | S\_IRWXG | S\_IRWXO)) == -1) {

puts("Unable to create semapthores");

exit(EXIT\_FAILURE);

}

int ctl\_sem\_binary = semctl(semid, SEM\_BINARY, SETVAL, 1);

int ctl\_sem\_empty = semctl(semid, SEM\_EMPTY , SETVAL, N);

int ctl\_sem\_full = semctl(semid, SEM\_FULL , SETVAL, 0);

if (ctl\_sem\_binary == -1 || ctl\_sem\_empty == -1 || ctl\_sem\_full == -1) {

puts("Cannot set controll semaphores");

exit(EXIT\_FAILURE);

}

make\_consumer(0, semid, 4);

make\_consumer(1, semid, 5);

make\_producer(0, semid, 97, 3);

make\_producer(1, semid, 98, 3);

make\_producer(2, semid, 99, 3);

int status;

for (int i = 0; i < 5; i++) {

wait(&status);

}

if (shmdt(shm\_attached\_address) == -1) { // Detach shared memory segment

puts("Can't detach shared memory segment");

exit(EXIT\_FAILURE);

}

exit(EXIT\_SUCCESS);

}

# Полученный результат:

# 

# 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.

# Листинг кода:

#include <stdio.h>

#include <stdlib.h> // rand

#include <unistd.h> // sleep

#include <signal.h>

#include <time.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <sys/ipc.h>

#include <sys/sem.h>

#include <sys/shm.h>

#include <sys/stat.h>

//объявление семафор

#define SEM\_AW 0

#define SEM\_WW 1

#define SEM\_AR 2

#define SEM\_WR 3

#define SEM\_N 4

#define INC 1

#define DEC -1

#define SLP 0

struct sembuf start\_read**[]** **=** **{**

**{** SEM\_WR**,** INC**,** 0 **},**

**{** SEM\_WW**,** SLP**,** 0 **},**

**{** SEM\_AW**,** SLP**,** 0 **},**

**{** SEM\_WR**,** DEC**,** 0 **},**

**{** SEM\_AR**,** INC**,** 0 **}**

**};**

struct sembuf stop\_read**[]** **=** **{**

**{** SEM\_AR**,** DEC**,** 0 **}**

**};**

struct sembuf start\_write**[]** **=** **{**

**{** SEM\_WW**,** INC**,** 0 **},**

**{** SEM\_AW**,** SLP**,** 0 **},**

**{** SEM\_AR**,** SLP**,** 0 **},**

**{** SEM\_WW**,** DEC**,** 0 **},**

**{** SEM\_AW**,** INC**,** 0 **}**

**};**

struct sembuf stop\_write**[]** **=** **{**

**{** SEM\_AW**,** DEC**,** 0 **}**

**};**

#define WRITERS 4

#define READERS 4

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

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

while (1) {

semop(semid, start\_write, SEM\_COUNT(start\_write));

(\*shm)++;

printf("Writer #%d write %d\n", num, \*shm);

semop(semid, stop\_write, SEM\_COUNT(stop\_write));

sleep(rand() % 5);

}

}

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

while (1) {

semop(semid, start\_read, SEM\_COUNT(start\_read));

printf("\t\t\tReader #%d read %d\n", num, \*shm);

semop(semid, stop\_read, SEM\_COUNT(stop\_read));

sleep(rand() % 5);

}

}

void make\_reader(int reader\_id, int sem\_id, int\* const shm\_buff) {

pid\_t pid = fork();

if (pid == -1) {

perror("Reader's fork error.\n");

perror("Terminating..\n");

kill(0, SIGKILL);

exit(EXIT\_FAILURE);

} else if (pid == 0) {

reader(sem\_id, shm\_buff, reader\_id);

exit(EXIT\_SUCCESS);

}

}

void make\_writer(int writer\_id, int sem\_id, int\* const shm\_buf) {

pid\_t pid = fork();

if (pid == -1) {

perror("Writer's fork error.\n");

perror("Terminating..\n");

kill(0, SIGKILL);

exit(EXIT\_FAILURE);

} else if (pid == 0) {

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

exit(EXIT\_SUCCESS);

}

}

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 | S\_IRWXU | S\_IRWXG | S\_IRWXO)) == -1) {

perror("!!! Unable to create a shared area.\n");

exit(EXIT\_FAILURE);

}

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

if (shm\_buf == (int\*)-1) {

perror("!!! Can't attach memory");

exit(EXIT\_FAILURE);

}

(\*shm\_buf) = 0;

int sem\_id;

if ((sem\_id = semget(IPC\_PRIVATE, SEM\_N, IPC\_CREAT | S\_IRWXU | S\_IRWXG | S\_IRWXO)) == -1)

{

perror("!!! Unable to create a semaphore.\n");

exit(EXIT\_FAILURE);

}

if (semctl(sem\_id, SEM\_AW, SETVAL, 0) == -1 ||

semctl(sem\_id, SEM\_WW, SETVAL, 0) == -1 ||

semctl(sem\_id, SEM\_AR, SETVAL, 0) == -1 ||

semctl(sem\_id, SEM\_WR, SETVAL, 0) == -1) {

perror( "!!! Can't set control semaphors." );

exit(EXIT\_FAILURE);

}

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

make\_writer(i, sem\_id, shm\_buf);

}

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

make\_reader(i, sem\_id, shm\_buf);

}

if (shmdt(shm\_buf) == -1) {

perror( "!!! Can't detach shared memory" );

exit(EXIT\_FAILURE);

}

int \*status = NULL;

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

wait(status);

}

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

perror( "!!! Can't free memory!" );

exit(EXIT\_FAILURE);

}

return 0;

}

# Полученный результат:

# 