## 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 процессов**

**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 <unistd.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/sem.h>

#include <sys/stat.h>

#include <sys/shm.h>

#include <stdlib.h>

#include <sys/wait.h>

#include <sys/mman.h>

#define RUN\_ACCESS 0

#define BUF\_EMPTY 1

#define BUF\_FULL 2

#define BUF\_SIZE 10

const int PROD\_MAX\_NUM **=** 3**;**

const int CONS\_MAX\_NUM **=** 5**;**

void pexit**(**const char **\*,** int err\_code**);**

void create\_processes**();**

void wait\_childs**();**

void clean\_resources**();**

unsigned short init\_value**[]** **=** **{**1**,** BUF\_SIZE**,** 0**};**

struct sembuf prod\_p**[**2**]** **=** **{{**BUF\_EMPTY**,** **-**1**,** 0**},** **{**RUN\_ACCESS**,** **-**1**,** 0**}};**

struct sembuf prod\_v**[**2**]** **=** **{{**BUF\_FULL**,** 1**,** 0**},** **{**RUN\_ACCESS**,** 1**,** 0**}};**

struct sembuf cons\_p**[**2**]** **=** **{{**BUF\_FULL**,** **-**1**,** 0**},** **{**RUN\_ACCESS**,** **-**1**,** 0**}};**

struct sembuf cons\_v**[**2**]** **=** **{{**BUF\_EMPTY**,** 1**,** 0**},** **{**RUN\_ACCESS**,** 1**,** 0**}};**

int shm\_id **=** **-**1**;**

int sem\_id **=** **-**1**;**

char **\***mem\_ptr **=** **NULL;**

char **\***buf **=** **NULL;**

char **\***proc\_n **=** **NULL;**

char **\***cons\_n **=** **NULL;**

char **\***value **=** **NULL;**

int childs **=** 0**;**

void producer**(**int id**)**

**{**

**while** **(**1**)** **{**

semop**(**sem\_id**,** prod\_p**,** 2**);**

buf**[\***proc\_n**]** **=** **\***value**;**

printf**(**"PRODUCER %d pos %d -----> produced %c\n"**,** id**,** **\***proc\_n**,** buf**[\***proc\_n**]);**

**if** **(\***value **==** 'z'**){**

**\***value **=** 'a'**;**

**}** **else** **{**

**\***value **=** **\***value **+** 1**;**

**}**

**if** **(\***proc\_n **==** BUF\_SIZE **-** 1**){**

**\***proc\_n **=** 0**;**

**}**

**else{**

**\***proc\_n **=** **\***proc\_n **+** 1**;**

**}**

semop**(**sem\_id**,** prod\_v**,** 2**);**

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

**}**

**}**

void consumer**(**int id**)**

**{**

**while** **(**1**)** **{**

semop**(**sem\_id**,** cons\_p**,** 2**);**

printf**(**"CONSUMER %d pos %d <----- cunsumed %c\n"**,** id**,** **\***cons\_n**,** buf**[\***cons\_n**]);**

**if** **(\***cons\_n **==** BUF\_SIZE **-** 1**){**

**\***cons\_n **=** 0**;**

**}**

**else{**

**\***cons\_n **=** **\***cons\_n **+** 1**;**

**}**

semop**(**sem\_id**,** cons\_v**,** 2**);**

sleep**(**10**);**

**}**

**}**

int main**(**void**)**

**{**

int perms **=** IPC\_CREAT **|** S\_IRWXU **|** S\_IRWXG **|** S\_IRWXO**;**

atexit**(**clean\_resources**);**

value **=** mmap**(NULL,** **sizeof(**char**),** PROT\_READ **|** PROT\_WRITE**,** MAP\_SHARED **|** MAP\_ANONYMOUS**,** **-**1**,** 0**);**

**\***value **=** 'a'**;**

**if** **((**shm\_id **=** shmget**(**IPC\_PRIVATE**,** **(**BUF\_SIZE **+** 2**)** **\*** **sizeof(**char**),** perms**))** **==** **-**1**)** **{**

pexit**(**"shmget"**,** 1**);**

**}**

**if** **((**mem\_ptr **=** shmat**(**shm\_id**,** **NULL,** 0**))** **==** **(**char **\*)** **-**1**)** **{**

pexit**(**"shmat"**,** 2**);**

**}**

proc\_n **=** mem\_ptr**;**

cons\_n **=** mem\_ptr **+** **sizeof(**char**);**

buf **=** mem\_ptr **+** 2 **\*** **sizeof(**char**);**

**\***proc\_n **=** **\***cons\_n **=** 0**;**

**if** **((**sem\_id **=** semget**(**IPC\_PRIVATE**,** 3**,** perms**))** **==** **-**1**)** **{**

pexit**(**"semget"**,** 4**);**

**}**

**if** **(**semctl**(**sem\_id**,** 0**,** SETALL**,** init\_value**)** **==** **-**1**)** **{**

pexit**(**"semctl"**,** 5**);**

**}**

create\_processes**();**

wait\_childs**();**

**return** 0**;**

**}**

void pexit**(**const char **\***msg**,** int err\_code**)**

**{**

perror**(**msg**);**

exit**(**err\_code**);**

**}**

void create\_processes**()**

**{**

**for** **(**int i **=** 0**;** i **<** PROD\_MAX\_NUM**;** i**++)** **{**

pid\_t pid **=** fork**();**

**if** **(**pid **<** 0**)**

pexit**(**"fork"**,** 1**);**

**if** **(!**pid**)** **{**

producer**(**i **+** 1**);**

**}** **else** **{**

childs**++;**

**}**

**}**

**for** **(**int i **=** 0**;** i **<** CONS\_MAX\_NUM**;** i**++)** **{**

pid\_t pid **=** fork**();**

**if** **(**pid **<** 0**)**

pexit**(**"fork"**,** 1**);**

**if** **(!**pid**)** **{**

consumer**(**i **+** 1**);**

**}** **else** **{**

childs**++;**

**}**

**}**

**}**

void wait\_childs**()**

**{**

**while** **(**childs**--)**

wait**(NULL);**

**}**

void clean\_resources**()**

**{**

**if** **(**mem\_ptr **!=** **(**void **\*)** **-**1**)** shmdt**(**mem\_ptr**);**

**if** **(**sem\_id **>=** 0**)** semctl**(**sem\_id**,** 0**,** IPC\_RMID**);**

**if** **(**shm\_id **>=** 0**)** shmctl**(**shm\_id**,** IPC\_RMID**,** 0**);**

**if** **(**value **!=** **NULL)** munmap**(**value**,** **sizeof(**int**));**

**}**

}

# 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. Участвуют в синхронизации параллельно исполняемых процессах.

#define SEM\_AW 0 //активный писатель

#define SEM\_WW 1 //ждущий писатель

#define SEM\_AR 2 // активный читатель

#define SEM\_WR 3 // активный читатель

#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:

# 