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»

О Т Ч Е Т

по лабораторной работе № 05

Название ***Взаимодействие параллельных процессов***

Дисциплина: ***Операционные системы***

Студент ***ИУ7И-56Б*** _____ **Нгуен Ф. С.**

(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель _____ **Рязанова Н. Ю.**

(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.

```c
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <sys/sem.h>
#include <sys/shm.h>
#include <time.h>
#include <unistd.h>
#include <sys/wait.h>

#define BIN_SEM   0
#define BUFFER_EMPTY 1
#define BUFFER_FULL  2

#define DEC -1
#define INC 1

#define PRODUCERS_COUNT 3
#define CONSUMERS_COUNT 3

#define PRODUCERS_DELAY 3
#define CONSUMERS_DELAY 2

struct sembuf producerP[2] = { {BUFFER_EMPTY, DEC, SEM_UNDO},
                               {BIN_SEM, DEC, SEM_UNDO}};

struct sembuf producerV[2] = { {BIN_SEM, INC, SEM_UNDO},
                               {BUFFER_FULL, INC, SEM_UNDO}};

struct sembuf consumerP[2] = { {BUFFER_FULL,  DEC, SEM_UNDO},
                               {BIN_SEM,   DEC, SEM_UNDO}};

struct sembuf consumerV[2] = { {BIN_SEM, INC, SEM_UNDO},
                               {BUFFER_EMPTY, INC, SEM_UNDO}};


#define N 26 /*'Z' - 'A' */

#define PERMS S_IRWXU | S_IRWXG | S_IRWXO

int semId = -1;
int shmId = -1;

int *shm = NULL;
int *shm_producer_count = NULL;
int *shm_consumer_count = NULL;
int *shm_symbol_now = NULL;

int randint(int a, int b)
{
    return a + rand() % (b - a + 1);
}
```

```
53.
54. int semrel(int semId)
55. {
56.     return semctl(semId, 0, IPC_RMID, 0);
57. }
58.
59. int shmrel(int semId)
60. {
61.     return shmctl(shmId, IPC_RMID, NULL);
62. }
63.
64. void forkChildren(const int n, void (*func)(const int))
65. {
66.     for (int i = 0; i < n; ++i)
67.     {
68.         const pid_t pid = fork();
69.         if (pid == -1)
70.         {
71.             perror("fork");
72.             exit(1);
73.         }
74.         else if (pid == 0)
75.         {
76.             if (func)
77.                 func(i);
78.             exit(1);
79.         }
80.     }
81. }
82.
83. void waitChildren(const int n)
84. {
85.     for (int i = 0; i < n; ++i)
86.     {
87.         int status;
88.         const pid_t child_pid = wait(&status);
89.         if (child_pid == -1)
90.         {
91.             perror("wait");
92.             exit(1);
93.         }
94.         if (WIFEXITED(status))
95.             printf("Process %d returns %d\n", child_pid, WEXITSTATUS(status));
96.         else if (WIFSIGNALED(status))
97.             printf("Process %d terminated with signal %d\n", child_pid, WTERMSIG(status));
98.         else if (WIFSTOPPED(status))
99.             printf("Process %d stopped due signal %d\n", child_pid, WSTOPSIG(status));
100.    }
101.}
102.
103.void producer(const int id)
104.{
105.    while(1)
106.    {
107.        sleep(randint(0, PRODUCERS_DELAY));
108.
109.        if (semop(semId, producerP, 2) == -1)
110.        {
111.            perror("semop");
112.            exit(1);
113.        }
114.
115.        /* положить в буфер */
116.        int symbol = 'A' + *shm_producer_count % ('Z' - 'A');
117.        *(shm + *shm_producer_count) = symbol;
118.        printf("Producer-%d (pid %d) produces %c\n", id, getpid(), symbol);
119.        (*shm_producer_count)++;
120.
121.        if (semop(semId, producerV, 2) == -1)
122.        {
123.            perror("semop");
124.            exit(1);
125.        }
```

```
126.    }
127.}
128.
129.void consumer(const int id)
130.{
131.    while(1)
132.    {
133.        sleep(randint(0, CONSUMERS_DELAY));
134.
135.        if (semop(semId, consumerP, 2) == -1)
136.        {
137.            perror("semop");
138.            exit(1);
139.        }
140.        /* взять из буфера */
141.        printf("\t\t\t\t\tConsumer %d (pid %d) consumes %c\n", id, getpid(), *(shm + *shm_consumer_coun
    t));
142.        (*shm_consumer_count)++;
143.
144.        if (semop(semId, consumerV, 2) == -1)
145.        {
146.            perror("semop");
147.            exit(1);
148.        }
149.    }
150.}
151.
152.void initSemaphore()
153.{
154.    /* два считающих семафора + один бинарный */
155.    semId = semget(IPC_PRIVATE, 3, IPC_CREAT | PERMS);
156.
157.    if (semId == -1)
158.    {
159.        perror("semget");
160.        exit(1);
161.    }
162.    /*количество заполненных ячеек равно 0*/
163.    /*Все ячейки буфера изначально пусты */
164.    if (semctl(semId, BIN_SEM,    SETVAL, 1) == -1 ||
165.        semctl(semId, BUFFER_EMPTY, SETVAL, N) == -1 ||
166.        semctl(semId, BUFFER_FULL,  SETVAL, 0) == -1)
167.    {
168.        perror("semctl");
169.        exit(1);
170.    }
171.}
172.
173.void createSharedMemory()
174.{
175.    // (N + 3) * sizeof(int) - kích thước
176.    //IPC_PRIVATE - tạo seg mới
177.    shmId = shmget(IPC_PRIVATE, (N + 2) * sizeof(int), IPC_CREAT | PERMS);
178.    if (shmId == -1)
179.    {
180.        perror("shmget");
181.        exit(1);
182.    }
183.    shm = shmat(shmId, 0, 0);
184.    if (shm == (void *) -1)
185.    {
186.        perror("shmat");
187.        exit(1);
188.    }
189.    shm_producer_count = shm;
190.    shm_consumer_count = shm + 1;
191.    *shm_producer_count = 0;
192.    *shm_consumer_count = 0;
193.    shm = shm + 2;
194.}
195.
196.int main()
197.{
```

```
198.    initSemaphore();
199.    createSharedMemory();
200.
201.    forkChildren(PRODUCERS_COUNT, producer);
202.    forkChildren(CONSUMERS_COUNT, consumer);
203.
204.    waitChildren(PRODUCERS_COUNT + CONSUMERS_COUNT);
205.
206.    shmrel(semId);
207.    semrel(semId);
208.}
```

```
nguyensang@K-virtual-machine:~/Desktop/OS2020/3$ ./1.exe
Producer-2 (pid 3735) produces A
                                        Consumer 0 (pid 3736) consumes A
Producer-0 (pid 3733) produces B
                                        Consumer 1 (pid 3737) consumes B
Producer-1 (pid 3734) produces C
                                        Consumer 2 (pid 3738) consumes C
Producer-2 (pid 3735) produces D
                                        Consumer 0 (pid 3736) consumes D
Producer-0 (pid 3733) produces E
                                        Consumer 1 (pid 3737) consumes E
Producer-1 (pid 3734) produces F
                                        Consumer 2 (pid 3738) consumes F
Producer-2 (pid 3735) produces G
                                        Consumer 0 (pid 3736) consumes G
Producer-0 (pid 3733) produces H
                                        Consumer 1 (pid 3737) consumes H
Producer-1 (pid 3734) produces I
                                        Consumer 2 (pid 3738) consumes I
Producer-2 (pid 3735) produces J
                                        Consumer 0 (pid 3736) consumes J
Producer-0 (pid 3733) produces K
                                        Consumer 1 (pid 3737) consumes K
Producer-1 (pid 3734) produces L
                                        Consumer 2 (pid 3738) consumes L
Producer-2 (pid 3735) produces M
Producer-0 (pid 3733) produces N
Producer-1 (pid 3734) produces O
                                        Consumer 0 (pid 3736) consumes M
                                        Consumer 1 (pid 3737) consumes N
                                        Consumer 2 (pid 3738) consumes O
Producer-2 (pid 3735) produces P
```

❖ 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.

```
#include <sys/shm.h>
#include <sys/sem.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <time.h>
#include <assert.h>
#include <sys/stat.h>

#define INC 1
#define DEC -1
#define CHK 0

#define PERMS S_IRWXU | S_IRWXG | S_IRWXO

#define SEM_ACTIVE_WRITERS 0
#define SEM_ACTIVE_READERS 1
#define SEM_WAITING_WRITERS 2
#define SEM_WAITING_READERS 3

#define READER_COUNT 5
#define WRITER_COUNT 3

struct sembuf CANREAD   [3] = { {SEM_ACTIVE_WRITERS, CHK, 0},
                                {SEM_WAITING_WRITERS, CHK, 0},
                                {SEM_WAITING_READERS, INC, 0}};

struct sembuf STARTREAD [2] = { {SEM_WAITING_READERS, DEC, 0},
                                {SEM_ACTIVE_READERS, INC, 0}};

struct sembuf STOPREAD  [1] = { {SEM_ACTIVE_READERS, DEC, 0}};

struct sembuf CANWRITE  [3] = { {SEM_ACTIVE_READERS,  CHK, 0},
                                {SEM_ACTIVE_WRITERS,  CHK, 0},
                                {SEM_WAITING_WRITERS, INC, 0}};

struct sembuf STARTWRITE[2] = { {SEM_ACTIVE_WRITERS,  INC, 0},
                                {SEM_WAITING_WRITERS, DEC, 0}};

struct sembuf stopwrite [1] = { {SEM_ACTIVE_WRITERS, DEC, 0}};

#define CANREAD_SIZE 3
#define STARTREAD_SIZE 2
#define STOPREAD_SIZE 1
#define CANWRITE_SIZE 3
#define STARTWRITE_SIZE 2
#define stopwrite_SIZE 1

int sem_id = -1;
int shm_id = -1;
int *shm = NULL;

int randint(int a, int b)
{
    return a + rand() % (b - a + 1);
}
```

```
60.
61. int initSemaphore()
62. {
63.     sem_id = semget(IPC_PRIVATE, 5, IPC_CREAT | PERMS);
64.     if (sem_id == -1)
65.     {
66.         perror("semget");
67.         exit(1);
68.     }
69.
70.     if (semctl(sem_id, SEM_ACTIVE_WRITERS, SETVAL, 0) == -1
    || semctl(sem_id, SEM_ACTIVE_READERS, SETVAL, 0) == -1 ||
71.         semctl(sem_id, SEM_WAITING_READERS, SETVAL, 0) == -1
    || semctl(sem_id, SEM_WAITING_WRITERS, SETVAL, 0) == -1)
72.     {
73.         perror("semctl");
74.         exit(1);
75.     }
76.
77.     return sem_id;
78. }
79.
80. void forkChildren(const int n, void (*func)(const int))
81. {
82.     for (int i = 0; i < n; ++i)
83.     {
84.         const pid_t pid = fork();
85.         if (pid == -1)
86.         {
87.             perror("Err: fork");
88.             exit(1);
89.         }
90.         else if (pid == 0)
91.         {
92.             if (func)
93.                 func(i);
94.             exit(1);
95.         }
96.     }
97. }
98.
99. void waitChildren(const int n)
100.{
101.    for (int i = 0; i < n; ++i)
102.    {
103.        int status;
104.        const pid_t child_pid = wait(&status);
105.        if (child_pid == -1)
106.        {
107.            perror("wait");
108.            exit(1);
109.        }
110.        if (WIFEXITED(status))
111.            printf("Process %d returns %d\n", child_pid, WEXITSTATUS(status));
112.        else if (WIFSIGNALED(status))
113.            printf("Process %d terminated with signal %d\n", child_pid, WTERMSIG(status));
114.        else if (WIFSTOPPED(status))
115.            printf("Process %d stopped due signal %d\n", child_pid, WSTOPSIG(status));
116.    }
117.}
118.
119.void createSharedMemory()
120.{
121.    shm_id = shmget(IPC_PRIVATE, sizeof(int), IPC_CREAT | PERMS);
122.    if (shm_id == -1)
123.    {
124.        perror("Err: shmget");
125.        exit(1);
126.    }
127.
128.    shm = shmat(shm_id, 0, 0);
129.    if (shm == (void *) -1)
130.    {
```

```
        perror("Err: shmat");
        exit(1);
    }
}
```

```
void start_read()
{
    if (semop(sem_id, CANREAD, CANREAD_SIZE) == -1)
    {
        perror("semop");
        exit(1);
    }

    if (semop(sem_id, STARTREAD, STARTREAD_SIZE) == -1)
    {
        perror("semop");
        exit(1);
    }
}

void stop_read()
{
    if (semop(sem_id, STOPREAD, STOPREAD_SIZE) == -1)
    {
        perror("semop");
        exit(1);
    }
}

void start_write()
{
    if (semop(sem_id, CANWRITE, CANWRITE_SIZE) == -1)
    {
        perror("semop");
        exit(1);
    }
    if (semop(sem_id, STARTWRITE, STARTWRITE_SIZE) == -1)
    {
        perror("semop");
        exit(1);
    }
}

void stop_write()
{
    if (semop(sem_id, stopwrite, stopwrite_SIZE) == -1)
    {
        perror("semop");
        exit(1);
    }
}

void reader(int id)
{
    while (1)
    {
        sleep(randint(1, 3));
        start_read();
        printf("\t\t\t\tReader-%d (pid %d) read %d\n", id, getpid(), *shm);
        stop_read();
    }
}

void writer(int id)
{
    while (1)
    {
        sleep(randint(1, 2));
        start_write();
        ++*shm;
        printf("Writer-%d (pid %d) wrote %d\n", id, getpid(), *shm);
        stop_write();
```

```
68.     }
69. }
70. 
71. int main()
72. {
73.     initSemaphore();
74.     createSharedMemory();
75. 
76.     forkChildren(WRITER_COUNT, writer);
77.     forkChildren(READER_COUNT, reader);
78. 
79.     waitChildren(WRITER_COUNT + READER_COUNT);
80. 
81.     shmctl(shm_id, IPC_RMID, NULL);
82.     semctl(sem_id, SEM_ACTIVE_WRITERS, IPC_RMID, 0);
83. }
```

```
nguyensang@K-virtual-machine:~/Desktop/OS2020/3$ ./2.exe
Writer-0 (pid 7392) wrote 1
Writer-1 (pid 7393) wrote 2
Writer-2 (pid 7394) wrote 3
                                Reader-0 (pid 7395) read 3
                                Reader-1 (pid 7396) read 3
                                Reader-2 (pid 7397) read 3
                                Reader-3 (pid 7398) read 3
Writer-1 (pid 7393) wrote 4
Writer-0 (pid 7392) wrote 5
                                Reader-4 (pid 7399) read 5
Writer-2 (pid 7394) wrote 6
Writer-2 (pid 7394) wrote 7
Writer-1 (pid 7393) wrote 8
Writer-0 (pid 7392) wrote 9
                                Reader-1 (pid 7396) read 9
                                Reader-0 (pid 7395) read 9
                                Reader-2 (pid 7397) read 9
                                Reader-3 (pid 7398) read 9
                                Reader-4 (pid 7399) read 9
Writer-2 (pid 7394) wrote 10
Writer-1 (pid 7393) wrote 11
Writer-0 (pid 7392) wrote 12
                                Reader-3 (pid 7398) read 12
                                Reader-0 (pid 7395) read 12
                                Reader-2 (pid 7397) read 12
                                Reader-1 (pid 7396) read 12
                                Reader-4 (pid 7399) read 12
                                Reader-2 (pid 7397) read 12
```