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 № 3

Название: Взаимоисключение при взаимодействии параллельных процессов. Семафоры и разделяемая память.

Дисциплина: Операционные системы

Студент ИУ7-52Б _____ Е.В. Брянская
(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 3
изводителей 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.

```
#define     N           10
#define     NUM_PR      3

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

struct sembuf before_putting_in_buf[2] =
{
    {SE, -1, SEM_UNDO},
    {SB, -1, SEM_UNDO}
};

struct sembuf after_putting_in_buf[2] =
{
    {SB, 1, SEM_UNDO},
    {SF, 1, SEM_UNDO}
};

struct sembuf before_taking_from_buf[2] =
{
    {SF, -1, SEM_UNDO},
    {SB, -1, SEM_UNDO}
};

struct sembuf after_taking_from_buf[2] =
{
    {SB, 1, SEM_UNDO},
    {SE, 1, SEM_UNDO}
};


int action_producer(int num_pr, int id_sem, int n, char* addr)
{
    int temp;
```

```
    while (1)
    {
        sleep(rand() % 4 + 1);

        temp = semop(id_sem, before_putting_in_buf, 2);
        if (temp == -1)
        {
            perror("semop error");
            return 6;
        }

        addr[addr[1]] = addr[2];
        printf(">> PRODUCER %d: put %c\n", num_pr + 1, addr[2]);
        addr[2]++;

        if (addr[2] > 'z')
            addr[2] = 'a';

        (addr[1])++;

        if (addr[1] > n - 1)
            addr[1] = 3;

        temp = semop(id_sem, after_putting_in_buf, 2);
        if (temp == -1)
        {
            perror("semop error");
            return 6;
        }
    }

    return 0;
}

int action_consumer(int num_cn, int cur_id, int n, char* addr)
{
    int temp;

    while (1)
    {
        sleep(rand() % 4 + 1);

        temp = semop(cur_id, before_taking_from_buf, 2);
        if (temp == -1)
        {
            perror("semop error");
            return 6;
        }

        printf(">> CONSUMER %d: took %c\n", num_cn + 1,addr[addr[0]]);
        addr[0]++;
```

```
        if (addr[0] > n - 1)
            addr[0] = 3;

        temp = semop(cur_id, after_taking_from_buf, 2);
        if (temp == -1)
        {
            perror("semop error");
            return 6;
        }
    }

    return 0;
}


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

    int perms = S_IRWXU | S_IRWXG | S_IRWXO;
    int se, sf, sb, res, status, temp_id1, temp_id2;
    pid_t producers[NUM_PR], consumers[NUM_PR];

    int id_shm = shmget(IPC_PRIVATE, (N + 3) * sizeof(char), IPC_CREAT
 | perms);
    if (id_shm == -1)
    {
        perror("shmget error");
        exit(1);
    }

    char* addr = (char*)shmat(id_shm, 0, 0);
    if (addr == (char*)-1)
    {
        perror("shmat error");
        exit(2);
    }

    addr[0] = (char)3;
    addr[1] = (char)3;
    addr[2] = 'a';

    int id_sem = semget(IPC_PRIVATE, 3, IPC_CREAT | perms);
    if (id_sem == -1)
    {
        perror("semget error");
        exit(3);
    }

    se = semctl(id_sem, SE, SETVAL, N);
```

```
sf = semctl(id_sem, SF, SETVAL, 0);
sb = semctl(id_sem, SB, SETVAL, 1);

if (se == -1 || sf == -1 || sb == -1)
{
    perror("semctl error");
    exit(4);
}

for (int i = 0; i < NUM_PR; i++)
{
    producers[i] = fork();
    if (producers[i] == -1)
    {
        perror("fork error");
        exit(5);
    }

    if (producers[i] == 0)
    {
        res = action_producer(i, id_sem, N + 3, addr);
        if (res)
            exit(res);
    }

    rand();

    consumers[i] = fork();
    if (consumers[i] == -1)
    {
        perror("fork error");
        exit(5);
    }

    if (consumers[i] == 0)
    {
        res = action_consumer(i, id_sem, N + 3, addr);
        if (res)
            exit(res);
    }

    rand();
}

for (int i = 0; i < NUM_PR; i++)
{
    temp_id1 = wait(&status);
    temp_id2 = wait(&status);

    if (temp_id1 == -1 || temp_id2 == -1)
    {
```

```
            perror("wait error");
            exit(7);
        }
    }

    if (shmctl(id_shm, IPC_RMID, NULL) == -1)
    {
        perror("shmctl error");
        exit(4);
    }

    if (shmctl(id_sem, 0, IPC_RMID) == -1)
    {
        perror("shmctl error");
        exit(4);
    }

    return 0;
}
```

Результат выполнения программы:

```
ekaterina@ekaterina-HP-
>> PRODUCER 3: put a
>> CONSUMER 1: took a
>> PRODUCER 1: put b
>> CONSUMER 3: took b
>> PRODUCER 3: put c
>> PRODUCER 1: put d
>> PRODUCER 2: put e
>> CONSUMER 2: took c
>> CONSUMER 2: took d
>> CONSUMER 3: took e
>> PRODUCER 3: put f
>> CONSUMER 1: took f
>> PRODUCER 3: put g
>> CONSUMER 2: took g
>> PRODUCER 1: put h
>> CONSUMER 3: took h
>> PRODUCER 3: put i
>> PRODUCER 2: put j
>> CONSUMER 3: took i
>> PRODUCER 2: put k
>> CONSUMER 2: took j
>> CONSUMER 1: took k
>> PRODUCER 2: put l
```

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.

```
#define     SWR     0
#define     SWW     1
#define     SAR     2
#define     SAW     3

#define     NUM_W   3
#define     NUM_R   7

struct sembuf can_write_act[3] =
{
    { SAR, 0, SEM_UNDO },
    { SAW, 0, SEM_UNDO },
    { SWW, 1, SEM_UNDO }
};

struct sembuf start_write_act[2] =
{
    { SAW, 1, SEM_UNDO },
    { SWW, -1, SEM_UNDO }
};

struct sembuf stop_write_act[1] =
{
    { SAW, -1, SEM_UNDO }
};

struct sembuf can_read_act[3] =
{
    { SAW, 0, SEM_UNDO },
    { SAR, 0, SEM_UNDO },
    { SWR, 1, SEM_UNDO }
};

struct sembuf start_read_act[2] =
{
    { SAR, 1, SEM_UNDO },
    { SWR, -1, SEM_UNDO }
};

struct sembuf stop_read_act[1] =
{
    { SAR, -1, SEM_UNDO }
```

```
};

void start_write(int id_sem)
{
    int temp = semop(id_sem, can_write_act, 3);
    if (temp == -1)
    {
        perror("semop error");
        exit(6);
    }

    temp = semop(id_sem, start_write_act, 2);
    if (temp == -1)
    {
        perror("semop error");
        exit(6);
    }
}

void stop_write(int id_sem)
{
    int temp = semop(id_sem, stop_write_act, 1);
    if (temp == -1)
    {
        perror("semop error");
        exit(6);
    }
}

void start_read(int id_sem)
{
    int temp = semop(id_sem, can_read_act, 3);
    if (temp == -1)
    {
        perror("semop error");
        exit(6);
    }

    temp = semop(id_sem, start_read_act, 2);
    if (temp == -1)
    {
        perror("semop error");
        exit(6);
    }
}
void stop_read(int id_sem)
{
    int temp = semop(id_sem, stop_read_act, 1);
    if (temp == -1)
    {
        perror("semop error");
```

```
            exit(6);
        }
    }


    void action_writer(int cur_id, int id_sem, int* addr)
    {
        sleep(rand() % 2 + 1);

        while (1)
        {
            start_write(id_sem);

            (*addr)++;
            printf(">> WRITER %d: wrote %d\n", cur_id, *addr);

            stop_write(id_sem);

            sleep(rand() % 2 + 1);
        }
    }

    void action_reader(int cur_id, int id_sem, int* addr)
    {
        sleep(rand() % 3 + 1);

        while (1)
        {
            start_read(id_sem);

            printf(">> READER %d: read %d\n", cur_id, *addr);

            stop_read(id_sem);

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

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

        int perms = S_IRWXU | S_IRWXG | S_IRWXO;
        int swr, sww, sar, saw, res;
        pid_t writers[NUM_W], readers[NUM_R];

        int id_sem = semget(IPC_PRIVATE, 4, IPC_CREAT | perms);
        if (id_sem == -1)
        {
            perror("semget error");
            exit(1);
```

```
}

swr = semctl(id_sem, SWR, SETVAL, 0);
sww = semctl(id_sem, SWW, SETVAL, 0);
sar = semctl(id_sem, SAR, SETVAL, 0);
saw = semctl(id_sem, SAW, SETVAL, 0);

if (swr == -1 || sww == -1 || sar == -1 || saw == -1)
{
    perror("semctl error");
    exit(2);
}

int id_shm = shmget(IPC_PRIVATE, 1*sizeof(int),IPC_CREAT | perms);
if (id_shm == -1)
{
    perror("shmget error");
    exit(3);
}

int* addr = (int*)shmat(id_shm, 0, 0);
if (addr == (int*)-1)
{
    perror("shmat error");
    exit(4);
}

*addr = -1;

for (int i = 0; i < NUM_W; i++)
{
    writers[i] = fork();
    if (writers[i] == -1)
    {
        perror("fork error");
        exit(5);
    }

    if (writers[i] == 0)
        action_writer(i, id_sem, addr);

    rand();
}

for (int i = 0; i < NUM_R; i++)
{
    readers[i] = fork();
    if (readers[i] == -1)
    {
        perror("fork error");
        exit(5);
```

```
            }

            if (readers[i] == 0)
                action_reader(i, id_sem, addr);

            rand();
        }

        int status, temp_id;
        for (int i = 0; i < NUM_R + NUM_W; i++)
        {
            temp_id = wait(&status);

            if (temp_id == -1)
            {
                perror("wait error");
                exit(7);
            }
        }

        if (shmctl(id_shm, IPC_RMID, NULL) == -1)
        {
            perror("shmctl error");
            exit(8);
        }

        if (shmctl(id_sem, 0, IPC_RMID) == -1)
        {
            perror("shmctl error");
            exit(8);
        }

        return 0;
}
```

Результат выполнения программы:

```
ekaterina@ekaterina-HP-
>> WRITER 0: wrote 0
>> WRITER 1: wrote 1
>> WRITER 2: wrote 2
>> READER 2: read 2
>> READER 0: read 2
>> READER 3: read 2
>> READER 6: read 2
>> WRITER 0: wrote 3
>> READER 1: read 3
>> READER 2: read 3
>> READER 4: read 3
>> WRITER 1: wrote 4
>> READER 5: read 4
>> READER 6: read 4
>> WRITER 0: wrote 5
>> WRITER 2: wrote 6
>> READER 1: read 6
>> READER 0: read 6
>> READER 3: read 6
>> READER 5: read 6
>> READER 1: read 6
>> READER 0: read 6
>> READER 4: read 6
>> READER 2: read 6
>> WRITER 1: wrote 7
>> READER 5: read 7
>> WRITER 0: wrote 8
>> WRITER 2: wrote 9
>> READER 3: read 9
>> READER 4: read 9
>> READER 0: read 9
>> READER 6: read 9
>> READER 1: read 9
>> WRITER 1: wrote 10
```