|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **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**

**Отчет**

|  |  |
| --- | --- |
| **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");

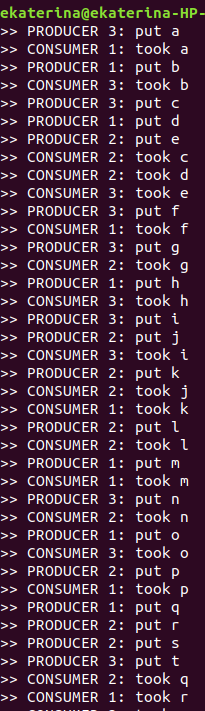
        exit(4);

    }

    return 0;

}

Результат выполнения программы:



**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 },

    { SAW, 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 },

    { SAR, 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 }

};

size\_t get\_len(struct sembuf arr[])

{

    return sizeof(arr) / sizeof((arr)[0]);

}

void start\_write(int id\_sem)

{

    int temp = semop(id\_sem, can\_write\_act, get\_len(can\_write\_act));

    if (temp == -1)

    {

        perror("semop error");

        exit(6);

    }

    temp = semop(id\_sem, start\_write\_act, get\_len(start\_write\_act));

    if (temp == -1)

    {

        perror("semop error");

        exit(6);

    }

}

void stop\_write(int id\_sem)

{

    int temp = semop(id\_sem, stop\_write\_act, get\_len(stop\_write\_act));

    if (temp == -1)

    {

        perror("semop error");

        exit(6);

    }

}

void start\_read(int id\_sem)

{

    int temp = semop(id\_sem, can\_read\_act, get\_len(can\_read\_act));

    if (temp == -1)

    {

        perror("semop error");

        exit(6);

    }

    temp = semop(id\_sem, start\_read\_act, get\_len(start\_read\_act));

    if (temp == -1)

    {

        perror("semop error");

        exit(6);

    }

}

void stop\_read(int id\_sem)

{

    int temp = semop(id\_sem, stop\_read\_act, get\_len(stop\_read\_act));

    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:

