



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»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №5
По курсу: "Операционные системы"

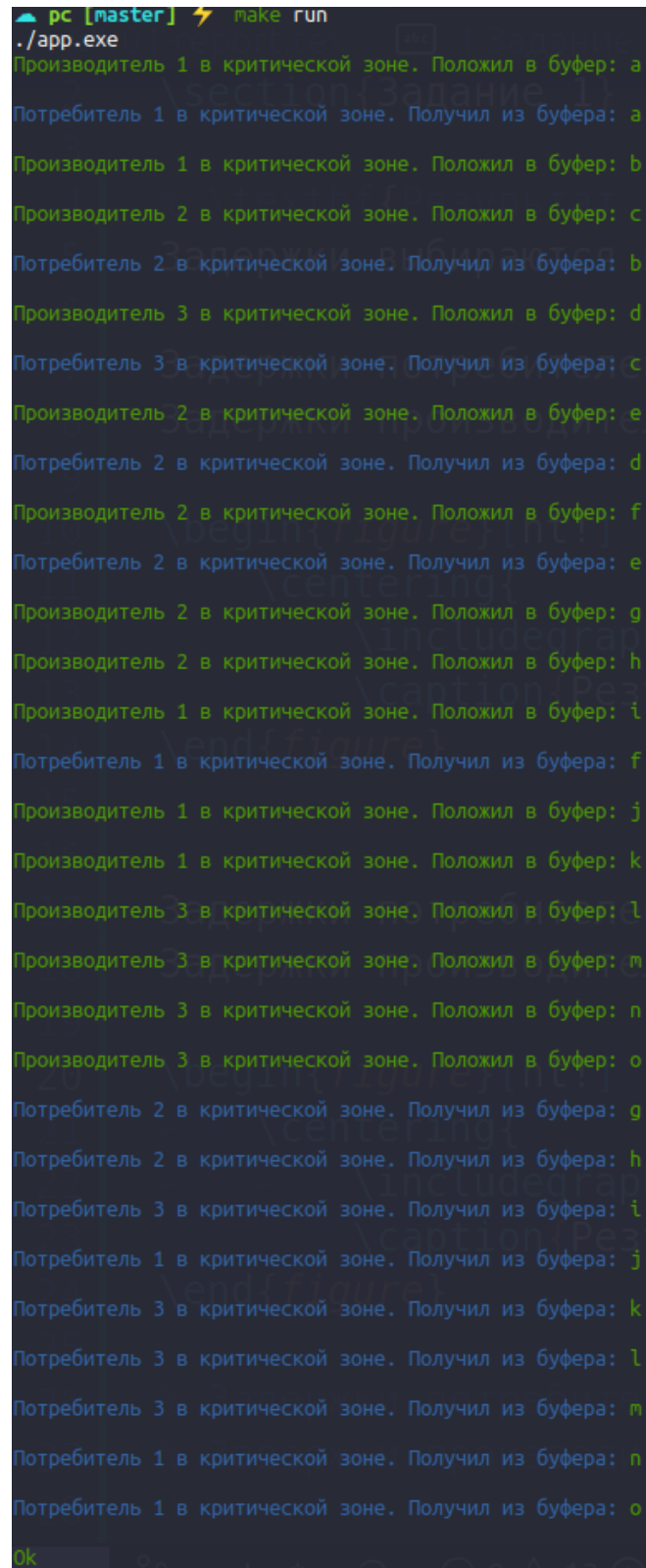
Студент _____ Сукочева Алис
Группа _____ ИУ7-53Б
Название предприятия _____ МГТУ им. Н. Э. Баумана, каф. ИУ7
Тема _____ Взаимодействие параллельных процессов.

Студент:	_____	Сукочева А.
	подпись, дата	Фамилия, И.О.
Преподаватель:	_____	Рязанова Н.Ю.
	подпись, дата	Фамилия, И. О.

Задание 1

Результат работы. Задержки выбираются произвольным образом из приведенных ниже.

Задержки потребителей: [0, 1, 2, 3] Задержки производителей: [0, 1]



```
pc [master] ⚡ make run
./app.exe
Производитель 1 в критической зоне. Положил в буфер: a
Потребитель 1 в критической зоне. Получил из буфера: a
Производитель 1 в критической зоне. Положил в буфер: b
Производитель 2 в критической зоне. Положил в буфер: c
Потребитель 2 в критической зоне. Получил из буфера: b
Производитель 3 в критической зоне. Положил в буфер: d
Потребитель 3 в критической зоне. Получил из буфера: c
Производитель 2 в критической зоне. Положил в буфер: e
Потребитель 2 в критической зоне. Получил из буфера: d
Производитель 2 в критической зоне. Положил в буфер: f
Потребитель 2 в критической зоне. Получил из буфера: e
Производитель 2 в критической зоне. Положил в буфер: g
Производитель 2 в критической зоне. Положил в буфер: h
Производитель 1 в критической зоне. Положил в буфер: i
Потребитель 1 в критической зоне. Получил из буфера: f
Производитель 1 в критической зоне. Положил в буфер: j
Производитель 1 в критической зоне. Положил в буфер: k
Производитель 3 в критической зоне. Положил в буфер: l
Производитель 3 в критической зоне. Положил в буфер: m
Производитель 3 в критической зоне. Положил в буфер: n
Потребитель 2 в критической зоне. Получил из буфера: g
Потребитель 2 в критической зоне. Получил из буфера: h
Потребитель 3 в критической зоне. Получил из буфера: i
Потребитель 1 в критической зоне. Получил из буфера: j
Потребитель 3 в критической зоне. Получил из буфера: k
Потребитель 3 в критической зоне. Получил из буфера: l
Потребитель 3 в критической зоне. Получил из буфера: m
Потребитель 1 в критической зоне. Получил из буфера: n
Потребитель 1 в критической зоне. Получил из буфера: o
ok
```

Рисунок 0.1 — Результат работы программы 1.

Задержки потребителей: [0, 1] Задержки производителей: [0, 1, 2, 3]

```
pc [master] ⚡ make run
./app.exe
Производитель 3 в критической зоне. Положил в буфер: a
Потребитель 1 в критической зоне. Получил из буфера: a
Производитель 2 в критической зоне. Положил в буфер: b
Потребитель 2 в критической зоне. Получил из буфера: b
Производитель 2 в критической зоне. Положил в буфер: c
Потребитель 3 в критической зоне. Получил из буфера: c
Производитель 3 в критической зоне. Положил в буфер: d
Потребитель 1 в критической зоне. Получил из буфера: d
Производитель 1 в критической зоне. Положил в буфер: e
Потребитель 1 в критической зоне. Получил из буфера: e
Производитель 3 в критической зоне. Положил в буфер: f
Потребитель 1 в критической зоне. Получил из буфера: f
Производитель 2 в критической зоне. Положил в буфер: g
Потребитель 2 в критической зоне. Получил из буфера: g
Производитель 1 в критической зоне. Положил в буфер: h
Потребитель 3 в критической зоне. Получил из буфера: h
Производитель 1 в критической зоне. Положил в буфер: i
Потребитель 1 в критической зоне. Получил из буфера: i
Производитель 1 в критической зоне. Положил в буфер: j
Потребитель 2 в критической зоне. Получил из буфера: j
Производитель 1 в критической зоне. Положил в буфер: k
Производитель 3 в критической зоне. Положил в буфер: l
Производитель 3 в критической зоне. Положил в буфер: m
Потребитель 3 в критической зоне. Получил из буфера: k
Потребитель 2 в критической зоне. Получил из буфера: l
Производитель 2 в критической зоне. Положил в буфер: n
Потребитель 2 в критической зоне. Получил из буфера: m
Производитель 2 в критической зоне. Положил в буфер: o
Потребитель 3 в критической зоне. Получил из буфера: n
Потребитель 3 в критической зоне. Получил из буфера: o
```

Рисунок 0.2 — Результат работы программы 1.

Задержки потребителей: [0, 1] Задержки производителей: [0, 1]

```

./app.exe
Производитель 1 в критической зоне. Положил в буфер: a
Потребитель 1 в критической зоне. Получил из буфера: a
Производитель 2 в критической зоне. Положил в буфер: b
Потребитель 2 в критической зоне. Получил из буфера: b
Производитель 3 в критической зоне. Положил в буфер: c
Потребитель 2 в критической зоне. Получил из буфера: c
Производитель 3 в критической зоне. Положил в буфер: d
Производитель 1 в критической зоне. Положил в буфер: e
Потребитель 1 в критической зоне. Получил из буфера: d
Потребитель 1 в критической зоне. Получил из буфера: e
Производитель 2 в критической зоне. Положил в буфер: f
Потребитель 3 в критической зоне. Получил из буфера: f
Производитель 2 в критической зоне. Положил в буфер: g
Потребитель 2 в критической зоне. Получил из буфера: g
Производитель 3 в критической зоне. Положил в буфер: h
Потребитель 3 в критической зоне. Получил из буфера: h
Производитель 3 в критической зоне. Положил в буфер: i
Производитель 3 в критической зоне. Положил в буфер: j
Производитель 1 в критической зоне. Положил в буфер: k
Потребитель 1 в критической зоне. Получил из буфера: i
Производитель 1 в критической зоне. Положил в буфер: l
Производитель 2 в критической зоне. Положил в буфер: m
Потребитель 1 в критической зоне. Получил из буфера: j
Потребитель 2 в критической зоне. Получил из буфера: k
Производитель 1 в критической зоне. Положил в буфер: n
Потребитель 3 в критической зоне. Получил из буфера: l
Производитель 2 в критической зоне. Положил в буфер: o
Потребитель 2 в критической зоне. Получил из буфера: m
Потребитель 3 в критической зоне. Получил из буфера: n
Потребитель 3 в критической зоне. Получил из буфера: o
ok

```

Рисунок 0.3 — Результат работы программы 1.

Листинг 1 — Главный файл main

```

1 struct sembuf InitValue[2] = {
2     {SB, 1, SEM_FLG}, // SB изначально установлен в 1.
3     {SE, N, SEM_FLG}  // SE изначально равно N.
4 };
5
6 int *consumer_pos = NULL;
7 int *producer_pos = NULL;
8 char *buffer = NULL;
9

```

```

10 const int shm_size = 2 * sizeof(int) + N * sizeof(char);
11
12 int main(void)
13 {
14     // Чтобы при повторном запуске новые рандомные числа были.
15     srand(time(NULL));
16
17     int semDescr;
18     int status;
19     int perms = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;
20     int *address = NULL;
21
22     // Создаем задержки.
23     Delay *delaysProducer = CreateRandomDelays(NUMBER_OF_WORKS,
24         PRODUCER_DELAY_TIME);
25     Delay *delaysConsumer = CreateRandomDelays(NUMBER_OF_WORKS,
26         CONSUMER_DELAY_TIME);
27
28     // shmget — создает новый разделяемый сегмент.
29     int shmid = shmget(IPC_PRIVATE, shm_size, perms);
30     if (shmid == ERROR_SHMGET)
31     {
32         perror("Не удалось создать разделяемый сегмент.\n");
33         return ERROR;
34     }
35
36     // Функция shmat() возвращает указатель на сегмент.
37     address = shmat(shmid, NULL, 0);
38     if (char address == -1)
39     {
40         perror("Не удалось получить указатель на сегмент.");
41         return ERROR;
42     }
43
44     // В начале разделяемой памяти хранится
45     // producer_pos и consumer_pos
46     // Начиная с buffer уже хранятся данные.
47     producer_pos = address;
48     *producer_pos = 0;
49     consumer_pos = address + sizeof(int);
50     *consumer_pos = 0;
51     buffer = (char *)(address + 2 * sizeof(int));
52
53     InitBuffer();
54
55     // Создаем новый набор, состоящий из 3 семафоров.
56     semDescr = semget(IPC_PRIVATE, SEM_COUNT, IPC_CREAT | perms);

```

```

55
56     if (semDescr == ERROR_SEMGET)
57     {
58         perror("Ошибка при создании набора семафоров.");
59         return ERROR;
60     }
61
62     // Задаем начальные значения семафоров.
63     if (semop(semDescr, InitValue, 2))
64     {
65         perror("Ошибка при попытке изменить семафор.");
66         return ERROR;
67     }
68
69     for (int i = 0; i < COUNT; i++)
70     {
71         CreateProducer(i + 1, semDescr, delaysProducer);
72         CreateConsumer(i + 1, semDescr, delaysConsumer);
73
74         // Обновляем задержки.
75         UpdateDelays(delaysProducer, PRODUCER_DELAY_TIME);
76         UpdateDelays(delaysConsumer, CONSUMER_DELAY_TIME);
77     }
78
79     for (int i = 0; i < COUNT_PRODUCER + COUNT_CONSUMER; i++)
80         wait(&status);
81
82     printf("%sOk\n", GREEN);
83
84     DestroyDelay(delaysProducer);
85     DestroyDelay(delaysConsumer);
86
87     if (shmdt(address) == -1)
88         perror("Ошибка при попытке отключить разделяемый сегмент от адресно-
89                го пространства процесса.");
90
91     return OK;
92 }

```

Листинг 2 — Файл для работы с задержками

```

1 Delay *CreateRandomDelays(int const count, const int delay_time)
2 {
3     Delay *delay = malloc(sizeof(Delay));
4
5     delay->delays = malloc(sizeof(int) * count);
6     delay->count = count - 1;

```

```

7
8     UpdateDelays(delay);
9
10    return delay;
11 }
12
13 void UpdateDelays(Delay *delay, const int delay_time)
14 {
15     for (int i = 0; i < delay->count; i++)
16         delay->delays[i] = rand() % DELAY_TIME;
17 }
18
19 int getDelay(Delay *delay)
20 {
21     if (delay->index > delay->count)
22         delay->index = 0;
23
24     return delay->delays[delay->index++];
25 }
26
27 void DestroyDelay(Delay *delay)
28 {
29     if (delay->delays)
30         free(delay->delays);
31     if (delay)
32         free(delay);
33 }

```

Листинг 3 — Потребитель

```

1 extern int *consumer_pos;
2 extern char *buffer;
3
4 // Потребитель.
5 struct sembuf ConsumerBegin[2] = {
6     {SF, P, SEM_FLG}, // Ожидает, что будет заполнена хотя бы одна ячейка б
        уфера.
7     {SB, P, SEM_FLG} // Ожидает, пока другой производитель или потребитель
        выйдет из критической зоны.
8 };
9
10 struct sembuf ConsumerEnd[2] = {
11     {SB, V, SEM_FLG}, // Освобождает критическую зону.
12     {SE, V, SEM_FLG} // Увеличивает кол-во пустых ячеек.
13 };
14
15 void ConsumerRunning(const int semId, const int consumerId, Delay *delays)

```

```

16 {
17     // Создаем случайные задержки.
18     sleep(getDelay(delays));
19     // printf("%s Задержка потребителя: %d\n", RED, getDelay(delays));
20
21     // Получаем доступ к критической зоне.
22     int rv = semop(semId, ConsumerBegin, 2); // rv = return value
23     if (rv == ERROR_SEMOP)
24     {
25         perror("Потребитель не может изменить значение семафора.\n");
26         exit(ERROR);
27     }
28
29     // Получить из буфера.
30     printf("%sПотребитель %d в критической зоне. Получил из буфера:
31         %s%c\n", BLUE, consumerId, GREEN, buffer[*consumer_pos]);
32
33     *consumer_pos = *consumer_pos + 1;
34
35     rv = semop(semId, ConsumerEnd, 2);
36     if (rv == ERROR_SEMOP)
37     {
38         perror("Потребитель не может изменить значение семафора.\n");
39         exit(ERROR);
40     }
41
42     puts("");
43 }
44
45 void CreateConsumer(const int consumerId, const int semId, Delay *delays)
46 {
47     pid_t childpid;
48     if ((childpid = fork()) == ERROR_FORK)
49     {
50         // Если при порождении процесса произошла ошибка.
51         perror("Ошибка при порождении процесса потребителя.");
52         exit(ERROR);
53     }
54     else if (!childpid) // childpid == 0
55     {
56         // Это процесс потомок.
57
58         // Каждый потребитель потребляет
59         // NUMBER_OF_WORKS товаров.
60         for (int i = 0; i < NUMBER_OF_WORKS; i++)
61             ConsumerRunning(semId, consumerId, delays);
62
63         exit(OK);
64     }
65 }

```



```
62     }
63 }
```

Листинг 4 — Производитель

```
1  extern int *producer_pos;
2  extern char *buffer;
3
4  // Производитель.
5  struct sembuf ProducerBegin[2] = {
6      {SE, P, SEM_FLG}, // Ожидает освобождения хотя бы одной ячейки буфера.
7      {SB, P, SEM_FLG}  // Ожидает, пока другой производитель или потребитель
                        // выйдет из критической зоны.
8  };
9  struct sembuf ProducerEnd[2] = {
10     {SB, V, SEM_FLG}, // Освобождает критическую зону.
11     {SF, V, SEM_FLG}  // Увеличивает кол-во заполненных ячеек.
12 };
13
14 void ProducerRunning(const int semId, const int producerId, Delay *delays)
15 {
16     // Создаем случайные задержки.
17     sleep(getDelay(delays));
18     // printf("%s Задержка потребителя: %d\n", RED, getDelay(delays));
19
20     // Получаем доступ к критической зоне.
21     int rv = semop(semId, ProducerBegin, 2); // rv = return value
22     if (rv == ERROR_SEMOP)
23     {
24         perror("Производитель не может изменить значение семафора.\n");
25         exit(ERROR);
26     }
27
28     // Положить в буфер.
29     printf("%sПроизводитель %d в критической зоне. Положил в буфер: %c\n",
30           YELLOW, producerId, ALPHABET[*producer_pos]);
31
32     buffer[*producer_pos] = ALPHABET[*producer_pos];
33     *producer_pos = *producer_pos + 1;
34
35     rv = semop(semId, ProducerEnd, 2);
36     if (rv == ERROR_SEMOP)
37     {
38         perror("Производитель не может изменить значение семафора.\n");
39         exit(ERROR);
40     }
41     puts("");
42 }
```

```

41 }
42
43 void CreateProducer(const int producerId, const int semId, Delay *delays)
44 {
45     pid_t childpid;
46     if ((childpid = fork()) == ERROR_FORK)
47     {
48         // Если при порождении процесса произошла ошибка.
49         perror("Ошибка при порождении процесса производителя.");
50         exit(ERROR);
51     }
52     else if (!childpid) // childpid == 0
53     {
54         // Это процесс потомок.
55
56         // Каждый производитель производит
57         // NUMBER_OF_WORKS товаров.
58         for (int i = 0; i < NUMBER_OF_WORKS; i++)
59             ProducerRunning(semId, producerId, delays);
60
61         exit(OK);
62     }
63 }

```

Листинг 5 — Файл с константами

```

1 #ifndef CONSTANTS_H
2
3 #define CONSTANTS_H
4
5 #define ALPHABET "abcdefghijklmnopqrstuvwxyz"
6 #define INIT_VALUE '0'
7
8 // Colors.
9 #define GREEN "\33[32m"
10 #define YELLOW "\33[33m"
11 #define BLUE "\33[34m"
12 #define RED "\33[31m"
13
14 // Errors
15 #define ERROR 1
16 #define ERROR_FORK -1
17 #define ERROR_PIPE -1
18 #define ERROR_SEMOP -1
19 #define ERROR_SEMGET -1
20 #define ERROR_SHMGET -1
21

```

```

22 // Каждый производитель производит 5 товаров.
23 // Каждый потребитель потребляет 5 товаров
24 #define NUMBER_OF_WORKS 5
25 // В программе создается 3 производителя =>
26 // 3 * 5 = 15 ячеек памяти потребуется.
27 #define N 15
28
29 #define OK 0
30
31 #define SEM_COUNT 3
32
33 // Для задержек.
34 #define CONSUMER_DELAY_TIME 1
35 #define PRODUCER_DELAY_TIME 3
36 #define DELAY_TIME 3
37
38 #define COUNT 3
39 #define COUNT_PRODUCER 3
40 #define COUNT_CONSUMER 3
41
42 // semaphore:
43 #define SF 0 // buffer full;
44 #define SE 1 // buffer empty;
45 #define SB 2 // binary.
46
47 // Операции над семафорами:
48 #define P -1 // Пропустить;
49 #define V 1 // Освободить.
50
51 #define SEM_FLG 0
52
53 #endif

```

Задание 2

Листинг 6 — Главный файл main

```

1 int *counter = NULL;
2
3 int main(void)
4 {
5     int semDescr;
6     int status;
7     int perms = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;
8
9     // shmget — создает новый разделяемый сегмент.

```

```

10 int shmids = shmget(IPC_PRIVATE, INT_SIZE, perms);
11 if (shmids == ERROR_SHMGET)
12 {
13     perror("Не удалось создать разделяемый сегмент.\n");
14     return ERROR;
15 }
16
17 // Функция shmat() возвращает указатель на сегмент
18 counter = shmat(shmids, NULL, 0);
19 if (char counter == -1)
20 {
21     perror("Не удалось получить указатель на сегмент.");
22     return ERROR;
23 }
24
25 *counter = 0;
26
27 // Создаем новый набор, состоящий из SEM_COUNT семафоров.
28 semDescr = semget(IPC_PRIVATE, SEM_COUNT, IPC_CREAT | perms);
29
30 if (semDescr == ERROR_SEMGET)
31 {
32     perror("Ошибка при создании набора семафоров.");
33     return ERROR;
34 }
35
36 for (int i = 0; i < NUMBER_READERS; i++)
37     CreateReader(semDescr, i + 1);
38
39 for (int i = 0; i < NUMBER_WRITERS; i++)
40     CreateWriter(semDescr, i + 1);
41
42 for (int i = 0; i < NUMBER_READERS + NUMBER_WRITERS; i++)
43     wait(&status);
44
45 if (shmdt(counter) == -1)
46     perror("Ошибка при попытке отключить разделяемый сегмент от адресно-
47     го пространства процесса.");
48
49 return OK;
50 }

```

Листинг 7 — Файл содержащий константы.

```

1 #ifndef CONSTANTS_H
2
3 #define CONSTANTS_H

```

```

4
5 #define ALPHABET "abcdefghijklmnopqrstuvwxyz"
6 #define INIT_VALUE '0'
7
8 // Colors.
9 #define GREEN "\33[32m"
10 #define YELLOW "\33[33m"
11 #define BLUE "\33[34m"
12 #define RED "\33[31m"
13
14 // Errors
15 #define ERROR 1
16 #define ERROR_FORK -1
17 #define ERROR_PIPE -1
18 #define ERROR_SEMOP -1
19 #define ERROR_SEMGET -1
20 #define ERROR_SHMGET -1
21
22 #define OK 0
23
24 #define TRUE 1
25 #define FALSE 0
26
27 #define INT_SIZE sizeof(int)
28
29 #define NUMBER_READERS 5
30 #define NUMBER_WRITERS 3
31
32 #define READER_SLEEP_TIME 1
33 #define WRITER_SLEEP_TIME 2
34
35 // semaphore:
36 #define R 0 // READER Кол-во активных читателей;
37 #define CR 1 // CAN_WRITE Читатель может читать;
38 #define CW 2 // CAN_WRITE - Писатель может записать;
39 #define WW 3 // WAIT_WRITERS - Кол-во ожидающий писателей, которые хотят за
    писать.
40
41 #define SEM_COUNT 4
42
43 // Операции над семафорами:
44 #define P -1 // Пропустить;
45 #define V 1 // Освободить.
46 #define S 0 // sleep.
47
48 #define SEM_FLG 0
49

```

```
50 #endif
```

Листинг 8 — Читатель

```
1 struct sembuf StartRead[3] = {
2     {WW, S, SEM_FLG}, // Пропускает всех ожидающих запись писателей.
3     {CR, S, SEM_FLG}, // Ждет, пока писатель допишет.
4     {R, V, SEM_FLG}   // Увеличивает кол-во активных читателей.
5 };
6
7 struct sembuf StopRead[1] = {
8     {R, P, SEM_FLG} // Уменьшает кол-во активных читателей.
9 };
10
11 extern int *counter;
12
13 void Reader(const int semId, const int readerId)
14 {
15     int rv = semop(semId, StartRead, 3); // rv = return value
16     if (rv == ERROR_SEMOP)
17     {
18         perror("Читатель не может изменить значение семафора.\n");
19         exit(ERROR);
20     }
21
22     printf("%sЧитатель %d прочитал: %d\n", GREEN, readerId, *counter);
23
24     rv = semop(semId, StopRead, 1);
25     if (rv == ERROR_SEMOP)
26     {
27         perror("Читатель не может изменить значение семафора.\n");
28         exit(ERROR);
29     }
30
31     sleep(READER_SLEEP_TIME);
32 }
33
34 void CreateReader(const int semId, const int readerId)
35 {
36     pid_t childpid;
37     if ((childpid = fork()) == ERROR_FORK)
38     {
39         perror("Ошибка при порождении читателя.");
40         exit(ERROR);
41     }
42     else if (!childpid)
43     {
```

```

44     // Это процесс потомок.
45     while (TRUE)
46         Reader(semId, readerId);
47     exit(OK);
48 }
49 }

```

Листинг 9 — Писатель

```

1 struct sembuf StartWrite[6] = {
2     {WW, V, SEM_FLG}, // Увеличивает кол-во ожидающий писателей.
3     {R, S, SEM_FLG}, // Ждет, пока все читатели дочитают.
4     {CW, S, SEM_FLG}, // Ждет, пока что другой писатель допишет.
5     {CW, V, SEM_FLG}, // Запрещает писать.
6     {CR, V, SEM_FLG}, // Запрещает читать.
7     {WW, P, SEM_FLG} // Уменьшает кол-во ожидающий писателей. Т.к. он уже
                        не ждет, а пишет
8 };
9
10 struct sembuf StopWrite[2] = {
11     {CR, P, SEM_FLG}, // Разрешает читать
12     {CW, P, SEM_FLG} // Разрешает писать.
13 };
14
15 extern int *counter;
16
17 void Writer(const int semId, const int writerId)
18 {
19     int rv = semop(semId, StartWrite, 6); // rv = return value
20     if (rv == ERROR_SEMOP)
21     {
22         perror("Писатель не может изменить значение семафора.\n");
23         exit(ERROR);
24     }
25
26     *counter = *counter + 1;
27     printf("%sПисатель %d записал: %d\n", YELLOW, writerId, *counter);
28
29     rv = semop(semId, StopWrite, 2);
30     if (rv == ERROR_SEMOP)
31     {
32         perror("Писатель не может изменить значение семафора.\n");
33         exit(ERROR);
34     }
35
36     sleep(WRITER_SLEEP_TIME);
37 }

```

```
38
39 void CreateWriter(const int semId, const int writerId)
40 {
41     pid_t childpid;
42     if ((childpid = fork()) == ERROR_FORK)
43     {
44         perror("Ошибка при порождении писателя.");
45         exit(ERROR);
46     }
47     else if (!childpid)
48     {
49         // Это процесс потомок.
50         while (TRUE)
51             Writer(semId, writerId);
52         exit(OK);
53     }
54 }
```



```

rw [master] ⚡ make run
./app.exe
Читатель 1 прочитал: 0
Читатель 3 прочитал: 0
Читатель 2 прочитал: 0
Читатель 4 прочитал: 0
Читатель 5 прочитал: 0
Писатель 1 записал: 1
Писатель 2 записал: 2
Писатель 3 записал: 3
Читатель 1 прочитал: 3
Читатель 2 прочитал: 3
Читатель 3 прочитал: 3
Читатель 4 прочитал: 3
Читатель 5 прочитал: 3
Читатель 2 прочитал: 3
Читатель 1 прочитал: 3
Читатель 5 прочитал: 3
Читатель 4 прочитал: 3
Читатель 3 прочитал: 3
Писатель 2 записал: 4
Писатель 1 записал: 5
Писатель 3 записал: 6
Читатель 2 прочитал: 6
Читатель 5 прочитал: 6
Читатель 1 прочитал: 6
Читатель 4 прочитал: 6
Читатель 3 прочитал: 6
Читатель 2 прочитал: 6
Читатель 5 прочитал: 6
Читатель 4 прочитал: 6
Читатель 3 прочитал: 6
Читатель 1 прочитал: 6
Писатель 2 записал: 7
Писатель 1 записал: 8
Писатель 3 записал: 9
Читатель 2 прочитал: 9
Читатель 5 прочитал: 9
Читатель 3 прочитал: 9
Читатель 4 прочитал: 9
Читатель 1 прочитал: 9
Писатель 2 записал: 10
Читатель 2 прочитал: 10
Читатель 5 прочитал: 10
Читатель 3 прочитал: 10
Читатель 4 прочитал: 10
Читатель 1 прочитал: 10
Писатель 1 записал: 11
Писатель 3 записал: 12
Читатель 2 прочитал: 12
Читатель 5 прочитал: 12
Читатель 1 прочитал: 12
Читатель 3 прочитал: 12
Читатель 4 прочитал: 12
Писатель 2 записал: 13
Писатель 1 записал: 14
Писатель 3 записал: 15
Читатель 4 прочитал: 15
Читатель 2 прочитал: 15
Читатель 5 прочитал: 15
Читатель 3 прочитал: 15
Читатель 1 прочитал: 15

```

Рисунок 0.4 — Результат работы программы 2.