



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 дисциплине "Операционные системы"

Тема Взаимодействие параллельных процессов

Студент Романов А.В.

Группа ИУ7-53Б

Оценка (баллы) \_\_\_\_\_

Преподаватели Рязанова Н.Ю.



# 1 | Задача «Производство-потребление»

```
Producer 3 write: a
Consumer 3 read: a
Producer 2 write: b
Consumer 2 read: b
Producer 1 write: c
Consumer 3 read: c
Producer 2 write: d
Consumer 1 read: d
Producer 3 write: e
Consumer 2 read: e
Producer 1 write: f
Consumer 3 read: f
Producer 2 write: g
Consumer 1 read: g
Producer 3 write: h
Consumer 2 read: h
Producer 2 write: i
Consumer 3 read: i
Producer 3 write: j
Consumer 2 read: j
Producer 1 write: k
Consumer 1 read: k
Producer 2 write: l
Consumer 3 read: l
Producer 3 write: m
Consumer 2 read: m
Producer 1 write: n
Consumer 1 read: n
Producer 3 write: o
Consumer 2 read: o
Producer 1 write: p
Producer 2 write: q
Consumer 3 read: p
Consumer 1 read: q
Producer 3 write: r
Producer 1 write: s
Consumer 1 read: r
Consumer 2 read: s
Producer 3 write: t
Producer 2 write: u
Consumer 3 read: t
Consumer 1 read: u
Producer 1 write: v
Consumer 3 read: v
Producer 2 write: w
Consumer 1 read: w
Producer 1 write: x
Consumer 2 read: x
```

Рис. 1.1: Демонстрация работы программы «Производство-потребление». Задержка потребителя: от 1 до 4, задержка производителя: от 1 до 4

```
Producer 1 write: a
Producer 2 write: b
Consumer 1 read: a
Consumer 2 read: b
Producer 3 write: c
Producer 1 write: d
Producer 3 write: e
Producer 2 write: f
Producer 2 write: g
Producer 1 write: h
Producer 3 write: i
Consumer 3 read: c
Producer 3 write: j
Consumer 1 read: d
Consumer 2 read: e
Producer 1 write: k
Producer 2 write: l
Consumer 3 read: f
Producer 2 write: m
Producer 3 write: n
Producer 1 write: o
Producer 3 write: p
Producer 2 write: q
Consumer 1 read: g
Consumer 2 read: h
Producer 1 write: r
Producer 2 write: s
Producer 3 write: t
Consumer 3 read: i
Producer 2 write: u
Producer 1 write: v
Consumer 3 read: j
Producer 3 write: w
Producer 1 write: x
Consumer 2 read: k
Consumer 1 read: l
Consumer 2 read: m
Consumer 3 read: n
Consumer 2 read: o
Consumer 3 read: p
Consumer 1 read: q
Consumer 3 read: r
Consumer 2 read: s
Consumer 3 read: t
Consumer 1 read: u
Consumer 2 read: v
Consumer 1 read: w
Consumer 1 read: x
```

Рис. 1.2: Демонстрация работы программы «Производство-потребление». Задержка потребителя: от 1 до 9, задержка производителя: от 1 до 4

```
Producer 2 write: a
Consumer 2 read: a
Producer 3 write: b
Consumer 1 read: b
Producer 1 write: c
Consumer 3 read: c
Producer 2 write: d
Consumer 1 read: d
Producer 3 write: e
Consumer 2 read: e
Producer 1 write: f
Consumer 3 read: f
Producer 2 write: g
Consumer 2 read: g
Producer 3 write: h
Consumer 1 read: h
Producer 1 write: i
Consumer 3 read: i
Producer 1 write: j
Consumer 1 read: j
Producer 3 write: k
Consumer 2 read: k
Producer 2 write: l
Consumer 3 read: l
Producer 3 write: m
Consumer 1 read: m
Producer 1 write: n
Consumer 3 read: n
Producer 1 write: o
Consumer 2 read: o
Producer 2 write: p
Consumer 1 read: p
Producer 3 write: q
Consumer 3 read: q
Producer 2 write: r
Consumer 1 read: r
Producer 1 write: s
Consumer 3 read: s
Producer 2 write: t
Consumer 2 read: t
Producer 3 write: u
Consumer 1 read: u
Producer 2 write: v
Consumer 3 read: v
Producer 1 write: w
Consumer 2 read: w
Producer 3 write: x
Consumer 2 read: x
```

Рис. 1.3: Демонстрация работы программы «Производство-потребление». Задержка потребителя: от 1 до 4, задержка производителя: от 1 до 9

## 1.1 Листинги кода

В листингах 1.1 - 1.4 представлены исходные коды решения задачи «Производство-потребление».

Листинг 1.1: Реализация очереди на основе циклического буфера

```
1 #include "buffer.h"
2
3 int init_buffer(cbuffer_t *const buf) {
4     if (!buf) {
5         perror("Error while initializing buffer.\n");
6         return BUF_ERR;
7     }
8
9     return OK;
10 }
11
12 int write_buffer(cbuffer_t *const buf, const char element) {
13     if (!buf) {
14         return BUF_ERR;
15     }
16
17     buf->buffer[buf->write_pos++] = element;
18     buf->write_pos %= N;
19
20     return OK;
21 }
22
23 int read_buffer(cbuffer_t *buf, char *const element) {
24     if (!buf) {
25         return BUF_ERR;
26     }
27
28     if (!element) {
29         return BUF_ERR;
30     }
31
32     *element = buf->buffer[buf->read_pos++];
33     buf->read_pos %= N;
34
35     return OK;
36 }
```

Листинг 1.2: Реализация «потребителей»

```
1 #include "consumer.h"
2
3 struct sembuf CONS_LOCK[] = {
4     { BUFF_FULL, -1, 0 },
5     { BIN_SEM, -1, 0 }
```

```

6 };
7
8 struct sembuf CONS_RELEASE[] = {
9     { BUFF_EMPTY, 1, 0 },
10    { BIN_SEM, 1, 0 }
11 };
12
13 int consumer_run(cbuffer_t *const buf, const int sid, const int consid) {
14     srand(time(NULL) + consid);
15
16     if (!buf) {
17         return BUF_ERR;
18     }
19
20     for (int i = 0; i < ITER_CNT; i++) {
21         int sleep_time = rand() % CONS_TIME_RANGE + CONS_TIME_START;
22         sleep(sleep_time);
23
24         if (-1 == semop(sid, CONS_LOCK, SEM_SIZE)) {
25
26             return CONS_LOCK_ERROR;
27         }
28
29         char symb;
30
31         if (-1 == read_buffer(buf, &symb)) {
32             return BUFF_READ_ERROR;
33         }
34
35         fprintf(stdout, "Consumer %d read: %c\n", consid + 1, symb);
36
37         if (-1 == semop(sid, CONS_RELEASE, SEM_SIZE)) {
38             return CONS_RELEASE_ERROR;
39         }
40     }
41
42     return OK;
43 }

```

Листинг 1.3: Реализация «производителей»

```

1 #include "producer.h"
2
3 struct sembuf PROD_LOCK[] = {
4     { BUFF_EMPTY, -1, 0 },
5     { BIN_SEM, -1, 0 }
6 };
7
8 struct sembuf PROD_RELEASE[] = {
9     { BUFF_FULL, 1, 0 },

```

```

10 { BIN_SEM, 1, 0 }
11 };
12
13 int producer_run(cbuffer_t *const buf, const int sid, const int prodid) {
14     srand(time(NULL) + prodid);
15
16     if (!buf) {
17         return BUF_ERR;
18     }
19
20     for (size_t i = 0; i < ITER_CNT; i++) {
21         int sleep_time = rand() % PROD_TIME_RANGE + PROD_TIME_START;
22         sleep(sleep_time);
23
24         if (-1 == semop(sid, PROD_LOCK, SEM_SIZE)) {
25             return PROD_LOCK_ERROR;
26         }
27
28         const char symb = (buf->write_pos % 26) + 'a';
29         if (-1 == write_buffer(buf, symb)) {
30             return BUFF_WRITE_ERROR;
31         }
32
33         fprintf(stdout, "Producer %lu write: %c\n", prodid + 1, symb);
34
35         if (-1 == semop(sid, PROD_RELEASE, SEM_SIZE)) {
36             return PROD_RELEASE_ERROR;
37         }
38     }
39
40     return OK;
41 }

```

Листинг 1.4: Главный файл программы

```

1 #include <stdio.h>
2 #include <sys/ipc.h>
3 #include <sys/shm.h>
4 #include <sys/stat.h>
5 #include <sys/sem.h>
6 #include <unistd.h>
7 #include <wait.h>
8
9 #include "buffer.h"
10 #include "consumer.h"
11 #include "producer.h"
12
13 #define PROD_CNT 3
14 #define CONS_CNT 3
15

```



```

16 #define PERMISSIONS S_IRWXU | S_IRWXG | S_IRWXO
17 #define SHMAT_ERR_RET (void *)-1
18
19 #define SEM_CNT 3
20 #define BIN_SEM 0
21 #define BUFF_FULL 1
22 #define BUFF_EMPTY 2
23
24 #define FREE 1
25
26 #define SHMGET_ERROR 1
27 #define SHMAT_ERROR 2
28 #define FORK_ERROR 3
29 #define WAIT_ERROR 4
30 #define SHUTDOWN_ERROR 5
31 #define SEMGET_ERROR 12
32
33 int main() {
34     setbuf(stdout, NULL);
35
36     int fd = shmget(IPC_PRIVATE, sizeof(cbuffer_t), PERMISSIONS | IPC_CREAT);
37     if (-1 == fd) {
38         perror("Error while creating shared memory.\n");
39         return SHMGET_ERROR;
40     }
41
42     cbuffer_t *buffer = shmat(fd, 0, 0);
43     if (SHMAT_ERR_RET == buffer) {
44         perror("Error while creating shmat.\n");
45         return SHMAT_ERROR;
46     }
47
48     if (BUF_ERR == init_buffer(buffer)) {
49         return BUF_ERR;
50     }
51
52     int sid = semget(IPC_PRIVATE, SEM_CNT, PERMISSIONS | IPC_CREAT);
53     if (-1 == sid) {
54         perror("Error while creating array of semaphores.\n");
55         return SEMGET_ERROR;
56     }
57
58     semctl(sid, BIN_SEM, SETVAL, FREE);
59     semctl(sid, BUFF_EMPTY, SETVAL, N);
60     semctl(sid, BUFF_FULL, SETVAL, 0);
61
62     for (size_t i = 0; i < PROD_CNT; i++) {
63         int child_pid = fork();
64
65         if (-1 == child_pid) {

```

```

66     perror("Error while fork (producer).");
67     return FORK_ERROR;
68 } else if (0 == child_pid) {
69     producer_run(buffer, sid, i);
70     return OK;
71 }
72 }
73
74 for (size_t i = 0; i < CONS_CNT; i++) {
75     int child_pid = fork();
76
77     if (-1 == child_pid) {
78         perror("Error while fork (consumer).");
79         return FORK_ERROR;
80     } else if (0 == child_pid) {
81         consumer_run(buffer, sid, i);
82         return OK;
83     }
84 }
85
86 for (size_t i = 0; i < PROD_CNT + CONS_CNT; i++) {
87     int statval;
88
89     if (-1 == wait(&statval)) {
90         perror("Error with child process.\n");
91         return WAIT_ERROR;
92     }
93
94     if (!WIFEXITED(statval)) {
95         fprintf(stderr, "Children process %lu terminated abnormally.", i);
96     }
97 }
98
99 if (-1 == shmdt((void *)buffer) || -1 == shmctl(fd, IPC_RMID, NULL) || -1
100    == semctl(sid, IPC_RMID, 0)) {
101     perror("Error while shutdown.\n");
102     return SHUTDOWN_ERROR;
103 }
104
105 return OK;
106 }

```



## 2 | Задача «Читатели-писатели»

### 2.1 Демонстрация работы программы

```
Reader 1 read: 0
Reader 2 read: 0
Reader 4 read: 0
Writer 1 write: 1
Writer 2 write: 2
Writer 3 write: 3
Reader 3 read: 3
Reader 5 read: 3
Reader 1 read: 3
Reader 3 read: 3
Writer 1 write: 4
Writer 2 write: 5
Writer 3 write: 6
Reader 2 read: 6
Reader 4 read: 6
Reader 5 read: 6
Writer 1 write: 7
Writer 2 write: 8
Writer 3 write: 9
Reader 1 read: 9
Reader 2 read: 9
Reader 4 read: 9
Reader 3 read: 9
Writer 1 write: 10
Writer 2 write: 11
Writer 3 write: 12
Reader 1 read: 12
Reader 3 read: 12
Reader 2 read: 12
Reader 5 read: 12
Reader 1 read: 12
Writer 1 write: 13
Writer 2 write: 14
Writer 3 write: 15
Reader 4 read: 15
Reader 5 read: 15
Reader 1 read: 15
Writer 2 write: 16
Writer 1 write: 17
Writer 3 write: 18
Reader 3 read: 18
Reader 5 read: 18
Reader 1 read: 18
Reader 2 read: 18
Reader 1 read: 18
Writer 1 write: 19
Writer 2 write: 20
Writer 3 write: 21
```

Рис. 2.1: Демонстрация работы программы «Читатели-писатели».

## 2.2 Листинги кода

В листингах 1.1 - 1.4 представленные исходные коды решения задачи «Читатели-писатели».

Листинг 2.1: Реализация «читателей» и «писателей»

```
1 #include "read_write.h"
2
3 struct sembuf READER_QUEUE[] = {
4     { READ_QUEUE, 1, 0 },
5     { ACTIVE_WRITER, 0, 0 },
6     { WRITE_QUEUE, 0, 0 },
7 };
8
9 struct sembuf READER_LOCK[] = {
10     { ACTIVE_READER, 1, 0 },
11     { READ_QUEUE, -1, 0 },
12 };
13
14 struct sembuf READER_RELEASE[] = {
15     { ACTIVE_READER, -1, 0 },
16 };
17
18 struct sembuf WRITER_LOCK[] = {
19     { ACTIVE_WRITER, 1, 0 },
20     { WRITE_QUEUE, -1, 0 },
21 };
22
23 struct sembuf WRITER_RELEASE[] = {
24     { ACTIVE_WRITER, -1, 0 },
25 };
26
27 struct sembuf WRITER_QUEUE[] = {
28     { WRITE_QUEUE, 1, 0 },
29     { ACTIVE_READER, 0, 0 },
30     { ACTIVE_WRITER, 0, 0 },
31 };
32
33 int start_read(int sid) {
34     return semop(sid, READER_QUEUE, 3) != -1 && semop(sid, READER_LOCK, 2) !=
35         -1;
36 }
37
38 int stop_read(int sid) {
39     return semop(sid, READER_RELEASE, 1) != -1;
40 }
41
42 int start_write(int sid) {
43     return semop(sid, WRITER_QUEUE, 3) != -1 && semop(sid, WRITER_LOCK, 2) !=
44         -1;
45 }
```

```

44
45 int stop_write(int sid) {
46     return semop(sid, WRITER_RELEASE, 1) != -1;
47 }
48
49 int reader_run(int *const shared_mem, const int sid, const int rid) {
50     srand(time(NULL) + rid);
51
52     if (!shared_mem) {
53         return SHRDMEM_PTR_ERROR;
54     }
55
56     for (size_t i = 0; i < ITER_CNT; i++) {
57         int sleep_time = rand() % TIME_RANGE + TIME_START;
58         sleep(sleep_time);
59
60         if (!start_read(sid)) {
61             return READ_LOCK_ERROR;
62         }
63
64         int readed = *shared_mem;
65         fprintf(stdout, "Reader %d read: %d\n", rid + 1, readed);
66
67         if (!stop_read(sid)) {
68             return READ_RELEASE_ERROR;
69         }
70     }
71
72     return OK;
73 }
74
75 int writer_run(int *const shared_mem, const int sid, const int wid) {
76     srand(time(NULL) + wid);
77
78     if (!shared_mem) {
79         return SHRDMEM_PTR_ERROR;
80     }
81
82     for (size_t i = 0; i < ITER_CNT; i++) {
83         int sleep_time = rand() % TIME_RANGE + TIME_START;
84         sleep(sleep_time);
85
86         if (!start_write(sid)) {
87             return WRITE_LOCK_ERROR;
88         }
89
90         int updated = ++(*shared_mem);
91         fprintf(stdout, "Writer %d write: %d\n", wid + 1, updated);
92
93         if (!stop_write(sid)) {

```

```

94     return WRITE_RELEASE_ERROR;
95 }
96 }
97
98 return OK;
99 }

```

Листинг 2.2: Главный файл программы

```

1 #include <stdio.h>
2 #include <sys/shm.h>
3 #include <sys/stat.h>
4 #include <unistd.h>
5 #include <sys/types.h>
6 #include <sys/ipc.h>
7 #include <sys/sem.h>
8 #include <wait.h>
9
10 #include "read_write.h"
11
12 #define PERMISSIONS S_IRWXU | S_IRWXG | S_IRWXO
13 #define SHMAT_ERR_RET (void *)-1
14
15 #define READERS_CNT 5
16 #define WRITERS_CNT 3
17 #define SEM_CNT 4
18
19 #define OK 0
20 #define SHMGET_ERROR 1
21 #define SHMAT_ERROR 2
22 #define SEMGET_ERROR 3
23 #define FORK_ERROR 4
24 #define WAIT_ERROR 5
25 #define SHUTDOWN_ERROR 6
26
27 int main() {
28     setbuf(stdout, NULL);
29
30     int fd = shmget(IPC_PRIVATE, sizeof(int), PERMISSIONS | IPC_CREAT);
31     if (-1 == fd) {
32         perror("Error while creating shared memory.\n");
33         return SHMGET_ERROR;
34     }
35
36     int *shared_mem_ptr = shmat(fd, 0, 0);
37     if (SHMAT_ERR_RET == shared_mem_ptr) {
38         perror("Error while creating shmat.\n");
39         return SHMAT_ERROR;
40     }
41 }

```

```

42  int sid = semget(IPC_PRIVATE, SEM_CNT, PERMISSIONS | IPC_CREAT);
43  if (-1 == sid) {
44      perror("Error while creating array of semaphores.\n");
45      return SEMGET_ERROR;
46  }
47
48  semctl(sid, ACTIVE_READER, SETVAL, 0);
49  semctl(sid, ACTIVE_WRITER, SETVAL, 0);
50  semctl(sid, WRITE_QUEUE, SETVAL, 0);
51  semctl(sid, READ_QUEUE, SETVAL, 0);
52
53  for (size_t i = 0; i < READERS_CNT; i++) {
54      int child_pid = fork();
55
56      if (-1 == child_pid) {
57          perror("Error while fork (reader).");
58          return FORK_ERROR;
59      } else if (0 == child_pid) {
60          reader_run(shared_mem_ptr, sid, i);
61          return OK;
62      }
63  }
64
65  for (size_t i = 0; i < WRITERS_CNT; i++) {
66      int child_pid = fork();
67
68      if (-1 == child_pid) {
69          perror("Error while fork (reader).");
70          return FORK_ERROR;
71      } else if (0 == child_pid) {
72          writer_run(shared_mem_ptr, sid, i);
73          return OK;
74      }
75  }
76
77  for (size_t i = 0; i < READERS_CNT + WRITERS_CNT; i++) {
78      int statval;
79
80      if (-1 == wait(&statval)) {
81          perror("Error with child process.\n");
82          return WAIT_ERROR;
83      }
84
85      if (!WIFEXITED(statval)) {
86          fprintf(stderr, "Children process %lu terminated abnormally.", i);
87      }
88  }
89
90  if (-1 == shmdt((void *)shared_mem_ptr) || -1 == shmctl(fd, IPC_RMID, NULL)
    || -1 == semctl(sid, IPC_RMID, 0)) {

```



```
91     perror("Error while shutdown.\n");
92     return SHUTDOWN_ERROR;
93 }
94
95 return OK;
96 }
```