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"

Тема Взаимодействие параллельных процессов

Студент Пересторонин П.Г.

Группа ИУ7-53Б

Преподаватель Рязанова Н. Ю.

Москва — 2020 г.

Оглавление

1 Задача «Производство-потребление»

1.1 Вывод программы

На рисунках 1.1 – 1.3 показан вывод программы, реализующей задачу «Производство-потребление» при разных значениях задержки потребителей и производителей. Примечание: для удобства **производители** обозначены зеленым, в то время как **потребители** — красным; в скобках указано время последнего сна.

```
Producer #2 write: a (slept 2s)
Consumer #0 read:  a (slept 1s)
Producer #1 write: b (slept 3s)
Producer #0 write: c (slept 4s)
Consumer #1 read:  b (slept 4s)
Consumer #2 read:  c (slept 4s)
Producer #2 write: d (slept 3s)
Consumer #0 read:  d (slept 2s)
Producer #2 write: e (slept 1s)
Consumer #0 read:  e (slept 1s)
Producer #1 write: f (slept 4s)
Producer #0 write: g (slept 3s)
Consumer #1 read:  f (slept 3s)
Consumer #2 read:  g (slept 4s)
Producer #0 write: h (slept 2s)
Consumer #1 read:  h (slept 3s)
Producer #2 write: i (slept 4s)
Consumer #0 read:  i (slept 4s)
Producer #1 write: j (slept 4s)
Consumer #1 read:  j (slept 1s)
Producer #0 write: k (slept 3s)
Producer #1 write: l (slept 1s)
Consumer #2 read:  k (slept 4s)
Producer #2 write: m (slept 3s)
Consumer #0 read:  l (slept 3s)
Producer #2 write: n (slept 1s)
Producer #0 write: o (slept 3s)
Producer #1 write: p (slept 3s)
Consumer #1 read:  m (slept 4s)
Consumer #0 read:  n (slept 2s)
Producer #2 write: q (slept 1s)
Consumer #2 read:  o (slept 4s)
Producer #0 write: r (slept 1s)
Producer #1 write: s (slept 1s)
Consumer #1 read:  p (slept 1s)
Consumer #0 read:  q (slept 3s)
Producer #1 write: t (slept 2s)
Consumer #2 read:  r (slept 3s)
Producer #0 write: u (slept 3s)
Producer #2 write: v (slept 4s)
Consumer #1 read:  s (slept 3s)
Consumer #0 read:  t (slept 2s)
Producer #1 write: w (slept 3s)
Consumer #1 read:  u (slept 3s)
Producer #0 write: x (slept 4s)
Consumer #2 read:  v (slept 4s)
Consumer #2 read:  w (slept 2s)
Consumer #2 read:  x (slept 3s)
```

Рис. 1.1: «Производство-потребление» при задержках: потребители $\in [0, 5)$, производители $\in [0, 5)$

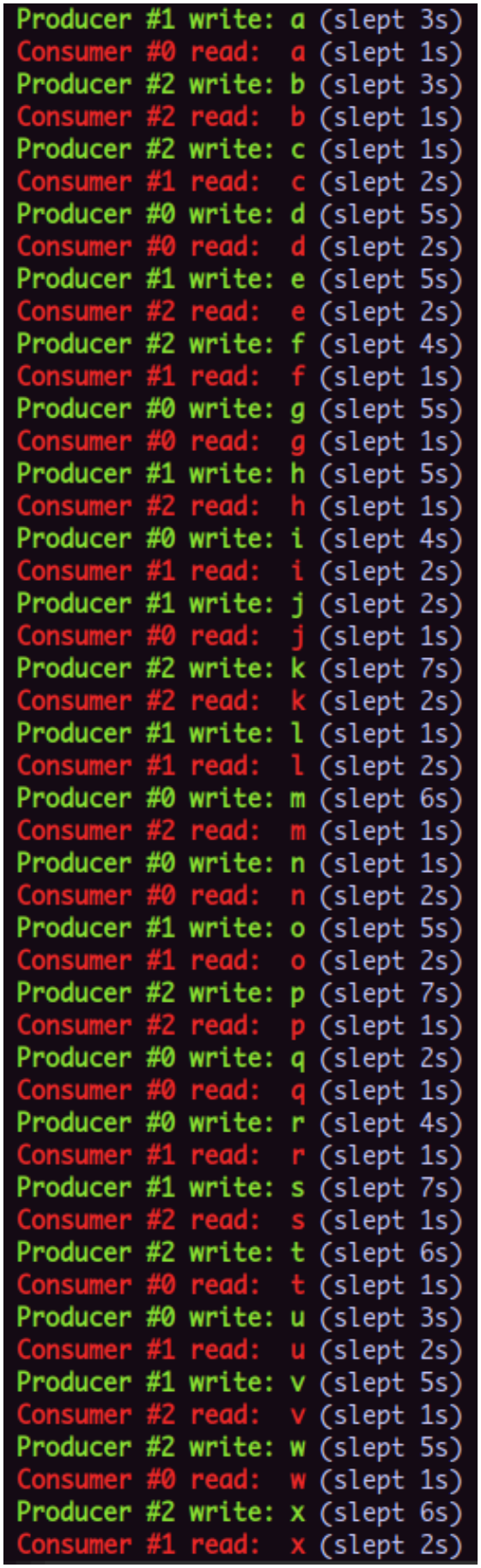

Рис. 1.2: «Производство-потребление» при задержках: потребители $\in [0, 3)$, производители $\in [0, 8)$

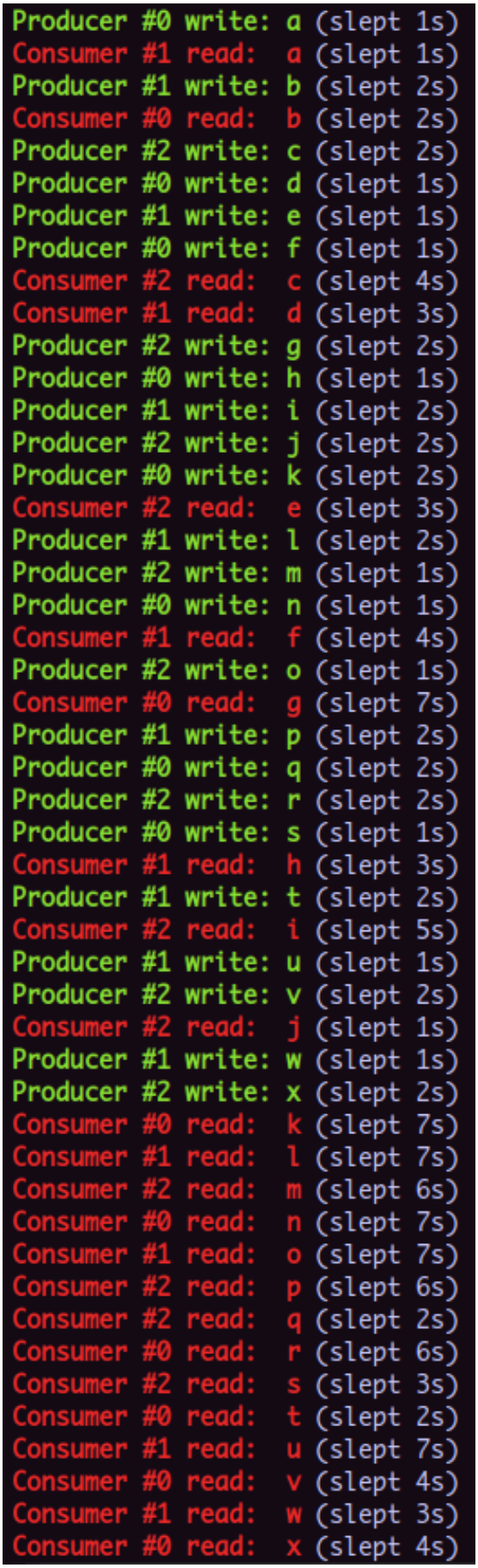

Рис. 1.3: «Производство-потребление» при задержках: потребители $\in [0, 8)$, производители $\in [0, 3)$

1.2 Листинги кода

В листингах 1.1 – 1.4 представлены исходные коды программы, реализующей задачу «Производство-потребление».

```
#include "buffer.h"

int init_cbuffer(CBuffer* const buffer) {
    if (!buffer)
        return -1;
    memset(buffer, 0, sizeof(CBuffer));
    return 0;
}

int write_cbuffer(CBuffer* const buffer, const char elem) {
    if (!buffer)
        return -1;
    buffer->data[buffer->wpos++] = elem;
    buffer->wpos %= N;
    return 0;
}

int read_cbuffer(CBuffer* const buffer, char* const dest) {
    if (!buffer)
        return -1;
    *dest = buffer->data[buffer->rpos++];
    buffer->rpos %= N;
    return 0;
}
```

Листинг 1.1: Реализация очереди, на основе циклического массива (буфера)

```
#include "runners.h"

struct sembuf PRODUCE_LOCK[2] = { { BUF_EMPTY, -1, 0 }, { BIN_SEM, -1, 0 } };
struct sembuf PRODUCE_RELEASE[2] = { { BUF_FULL, 1, 0 }, { BIN_SEM, 1, 0 } };

struct sembuf CONSUME_LOCK[2] = { { BUF_FULL, -1, 0 }, { BIN_SEM, -1, 0 } };
struct sembuf CONSUME_RELEASE[2] = { { BUF_EMPTY, 1, 0 }, { BIN_SEM, 1, 0 } };

int run_producer(CBuffer* const buffer, const int sid, const int prod_id) {
    if (!buffer)
        return -1;

    srand(time(NULL) + prod_id);

```

```
15      int sleep_time;
16      char ch;
17
18      for (int i = 0; i < ITERATIONS_AMOUNT; i++) {
19          sleep_time = rand() % PTIME_RANGE + PTIME_FROM;
20          sleep(sleep_time);
21
22          if (semop(sid, PRODUCE_LOCK, 2) == -1) {
23              perror("Something␣went␣wrong␣with␣produce␣lock!");
24              exit(PRODUCE_LOCK_FAILED);
25          }
26          // critical section
27          ch = 'a' + (char)(buffer->wpos % 26);
28          if (write_cbuffer(buffer, ch) == -1) {
29              perror("Something␣went␣wrong␣with␣buffer␣writing!");
30              return BUFFER_WRITE_FAILED;
31          }
32          printf("␣\e[1;32mProducer␣#%d␣write:␣%c\e[0m␣(slept␣%ds)\n", prod_id, ch,
                sleep_time);
33          // critical section ends
34          if (semop(sid, PRODUCE_RELEASE, 2) == -1) {
35              perror("Something␣went␣wrong␣with␣produce␣release!");
36              exit(PRODUCE_RELEASE_FAILED);
37          }
38
39      }
40      return 0;
41  }
42
43  int run_consumer(CBuffer* const buffer, const int sid, const int cons_id) {
44      if (!buffer)
45          return -1;
46
47      srand(time(NULL) + cons_id + PRODUCERS_AMOUNT);
48
49      int sleep_time;
50      char ch;
51
52      for (int i = 0; i < ITERATIONS_AMOUNT; i++) {
53          sleep_time = rand() % CTIME_RANGE + CTIME_FROM;
54          sleep(sleep_time);
55
56          if (semop(sid, CONSUME_LOCK, 2) == -1) {
57              perror("Something␣went␣wrong␣with␣consume␣lock!");
58              exit(CONSUME_LOCK_FAILED);
59          }
60          // critical section
61          if (read_cbuffer(buffer, &ch) == -1) {
```

```
            perror("Something␣went␣wrong␣with␣buffer␣reading!");
            return BUFFER_READ_FAILED;
        }
        printf("␣\e[1;31mConsumer␣#%d␣read:␣␣%c\e[0m␣(slept␣%ds)\n", cons_id, ch,
            sleep_time);
        // critical section ends
        if (semop(sid, CONSUME_RELEASE, 2) == -1) {
            perror("Something␣went␣wrong␣with␣write␣release!");
            exit(CONSUME_RELEASE_FAILED);
        }

    }
    return 0;
}
```

Листинг 1.2: Реализация “производителей” и “потребителей”

```
#ifndef __CONSTANTS__
#define __CONSTANTS__

#define PERMS S_IRWXU | S_IRWXG | S_IRWXO
#define KEY IPC_PRIVATE
#define N 64
#define ITERATIONS_AMOUNT 8
#define CONSUMERS_AMOUNT 3
#define PRODUCERS_AMOUNT 3

#define SEMS_AMOUNT 3
#define BIN_SEM 0
#define BUF_FULL 1
#define BUF_EMPTY 2

#define FREE 1

#define SHMGET_FAILED 1
#define SHMAT_FAILED 2
#define FORK_FAILED 3
#define PRODUCE_LOCK_FAILED 4
#define PRODUCE_RELEASE_FAILED 5
#define CONSUME_LOCK_FAILED 6
#define CONSUME_RELEASE_FAILED 7
#define BUFFER_WRITE_FAILED 8
#define BUFFER_READ_FAILED 9
#define WAIT_FAILED 10
#define SHUTDOWN_FAILED 11
#define SEMGET_FAILED 12

#define TRUE 1
#define FALSE 0
```

```
33
34 // Time [TIME_FROM, TIME_FROM + TIME_RANGE)
35
36 #define PTIME_FROM 1
37 #define PTIME_RANGE 2
38 #define CTIME_FROM 1
39 #define CTIME_RANGE 7
40
41 #endif // __CONSTANTS__
```

Листинг 1.3: Файл с константными значениями

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <sys/ipc.h>
5  #include <sys/sem.h>
6  #include <sys/shm.h>
7  #include <sys/stat.h>
8  #include <sys/types.h>
9  #include <unistd.h>
10 #include <time.h>
11 #include <wait.h>
12
13 #include "constants.h"
14 #include "buffer.h"
15 #include "runners.h"
16
17 int main(void) {
18     setbuf(stdout, NULL);
19     int fd = shmget(KEY, sizeof(CBuffer), IPC_CREAT | PERMS);
20     if (fd == -1) {
21         perror("Failed to create shared memory!");
22         return SHMGET_FAILED;
23     }
24
25     CBuffer *buffer;
26     if ((buffer = (CBuffer*)shmat(fd, 0, 0)) == (void*)-1) {
27         perror("Failed to shmat!");
28         return SHMAT_FAILED;
29     }
30
31     if (init_cbuffer(buffer) == -1) {
32         perror("Failed to init buffer!");
33         return SHMAT_FAILED;
34     }
35
36     int sid = semget(KEY, SEMS_AMOUNT, IPC_CREAT | PERMS);
37     if (sid == -1) {
```

```
38          perror("Can't␣create␣array␣of␣semaphores!");
39          return SEMGET_FAILED;
40      }
41      semctl(sid, BIN_SEM, SETVAL, FREE);
42      semctl(sid, BUF_EMPTY, SETVAL, N);
43      semctl(sid, BUF_FULL, SETVAL, 0);
44
45      int child_pid;
46      for (size_t i = 0; i < PRODUCERS_AMOUNT; i++) {
47          switch ((child_pid = fork())) {
48          case -1:
49              perror("Failed␣to␣fork␣for␣producer!");
50              exit(FORK_FAILED);
51              break;
52          case 0:
53              run_producer(buffer, sid, i);
54              return 0;
55          }
56      }
57
58      for (int i = 0; i < CONSUMERS_AMOUNT; i++) {
59          switch ((child_pid = fork())) {
60          case -1:
61              perror("Failed␣to␣fork␣for␣consumer!");
62              exit(FORK_FAILED);
63              break;
64          case 0:
65              run_consumer(buffer, sid, i);
66              return 0;
67          }
68      }
69
70      for (size_t i = 0; i < CONSUMERS_AMOUNT + PRODUCERS_AMOUNT; i++) {
71          int status;
72          if (wait(&status) == -1) {
73              perror("Something␣wrong␣with␣children␣waiting!");
74              exit(WAIT_FAILED);
75          }
76          if (!WIFEXITED(status))
77              printf("One␣of␣children␣terminated␣abnormally\n");
78      }
79
80      if (shmdt((void*)buffer) == -1 || shmctl(fd, IPC_RMID, NULL) == -1 || semctl(sid,
            IPC_RMID, 0) == -1) {
81          perror("Something␣went␣wrong␣during␣shutdown!");
82          return SHUTDOWN_FAILED;
83      }
84
```

```
85      return 0;
86  }
```

Листинг 1.4: Главный файл программы

2 Задача «Читатели-Писатели»

2.1 Вывод программы

На рисунке 2.1 показан вывод программы, реализующей задачу «Читатели-писатели». Примечание: для удобства **писатели** обозначены зеленым, в то время как **читатели** — красным; в скобках указано время последнего сна.

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

Рис. 2.1: «Читатели-писатели»

2.2 Листинги кода

В листингах 2.1 – 2.3 представлены исходные коды программы, реализующей задачу «Производство-потребление».

```
#include "io_objects.h"

struct sembuf READER_QUEUE[] = {
    { READ_QUEUE, 1, 0 }, // add member to read queue
    { ACT_WRTR, 0, 0 }, // No active writers
    { WRITE_QUEUE, 0, 0 }, // wait every member of write queue
};

struct sembuf READER_LOCK[] = {
    { ACT_RDRS, 1, 0 }, // inc active readers
    { READ_QUEUE, -1, 0 }, // leave queue (-1 in readers queue)
};
struct sembuf READER_RELEASE[] = {
    { ACT_RDRS, -1, 0 }, // -1 active reader
};

struct sembuf WRITER_QUEUE[] = {
    { WRITE_QUEUE, 1, 0 }, // add member to write queue
    { ACT_RDRS, 0, 0 }, // No active readers
    { ACT_WRTR, 0, 0 }, // No active writers
};

struct sembuf WRITER_LOCK[] = {
    { ACT_WRTR, 1, 0 }, // inc active writers
    { WRITE_QUEUE, -1, 0 }, // leave queue (-1 in writers queue)
};

struct sembuf WRITER_RELEASE[] = {
    { ACT_WRTR, -1, 0 }, // -1 active writer
};

int start_read(int sid) {
    return semop(sid, READER_QUEUE, 3) != -1 && semop(sid, READER_LOCK, 2) != -1;
}

int stop_read(int sid) {
    return semop(sid, READER_RELEASE, 1) != -1;
}

int run_reader(int* const shared_counter, const int sid, const int reader_id)
{
    if (!shared_counter)
```

```
43          return -1;
44
45      srand(time(NULL) + reader_id);
46
47      int sleep_time;
48      for (int i = 0; i < ITERATIONS_AMOUNT; i++) {
49          sleep_time = rand() % TIME_RANGE + TIME_FROM;
50          sleep(sleep_time);
51
52          if (!start_read(sid)) {
53              perror("Something␣went␣wrong␣with␣start_read!");
54              exit(READER_LOCK_FAILED);
55          }
56          // critical section
57          int val = *shared_counter;
58          printf("␣\e[1;31mReader␣#%d␣read:␣␣%5d\e[0m␣(slept␣%ds)\n", reader_id, val,
                sleep_time);
59          // critical section ends
60          if (!stop_read(sid)) {
61              perror("Something␣went␣wrong␣with␣stop_read!");
62              exit(READER_RELEASE_FAILED);
63          }
64      }
65      return 0;
66  }
67
68  int start_write(int sid) {
69      return semop(sid, WRITER_QUEUE, 3) != -1 && semop(sid, WRITER_LOCK, 2) != -1;
70  }
71
72  int stop_write(int sid) {
73      return semop(sid, WRITER_RELEASE, 1) != -1;
74  }
75
76  int run_writer(int* const shared_counter, const int sid, const int writer_id)
77  {
78      if (!shared_counter)
79          return -1;
80
81      srand(time(NULL) + writer_id + READERS_AMOUNT);
82
83      int sleep_time;
84      for (int i = 0; i < ITERATIONS_AMOUNT; i++) {
85          sleep_time = rand() % TIME_RANGE + TIME_FROM;
86          sleep(sleep_time);
87
88          if (!start_write(sid)) {
89              perror("Something␣went␣wrong␣with␣start_write!");
```

```
            exit(WRITER_LOCK_FAILED);
        }
        // critical section
        int val = ++(*shared_counter);
        printf("␣\e[1;32mWriter␣#%d␣write:␣%5d\e[0m␣(slept␣%ds)\n", writer_id, val,
            sleep_time);
        // critical section ends
        if (!stop_write(sid)) {
            perror("Something␣went␣wrong␣with␣stop_write!");
            exit(WRITER_RELEASE_FAILED);
        }
    }
    return 0;
}
```

Листинг 2.1: Реализация “читателей” и “писателей”

```
#ifndef __CONSTANTS__
#define __CONSTANTS__

#define PERMS S_IRWXU | S_IRWXG | S_IRWXO
#define KEY IPC_PRIVATE
#define ITERATIONS_AMOUNT 50
#define WRITERS_AMOUNT 3
#define READERS_AMOUNT 5

#define SEMS_AMOUNT 4
#define ACT_RDRS 0
#define ACT_WRTR 1
#define READ_QUEUE 2
#define WRITE_QUEUE 3

#define FREE 1

#define SHMGET_FAILED 1
#define SHMAT_FAILED 2
#define FORK_FAILED 3
#define READER_LOCK_FAILED 4
#define READER_RELEASE_FAILED 5
#define WRITER_LOCK_FAILED 6
#define WRITER_RELEASE_FAILED 7
#define WRITE_FAILED 8
#define READ_FAILED 8
#define WAIT_FAILED 10
#define SHUTDOWN_FAILED 11
#define SEMGET_FAILED 12

#define TRUE 1
#define FALSE 0
```

```
33
34 // Time [TIME_FROM, TIME_FROM + TIME_RANGE)
35 #define TIME_FROM 1
36 #define TIME_RANGE 5
37
38 #endif // __CONSTANTS__
```

Листинг 2.2: Файл с константными значениями

```
1  #include <sys/shm.h>
2  #include <wait.h>
3  #include <sys/stat.h>
4
5  #include "constants.h"
6  #include "io_objects.h"
7
8  int main(void) {
9      setbuf(stdout, NULL);
10     int fd = shmget(KEY, sizeof(int), IPC_CREAT | PERMS);
11     if (fd == -1) {
12         perror("Failed to create shared memory!");
13         return SHMGET_FAILED;
14     }
15
16     int* shared_counter;
17     if ((shared_counter = (int*)shmat(fd, 0, 0)) == (void*)-1) {
18         perror("Failed to shmat!");
19         return SHMAT_FAILED;
20     }
21
22     int sid = semget(KEY, SEMS_AMOUNT, IPC_CREAT | PERMS);
23     if (sid == -1) {
24         perror("Can't create array of semaphores!");
25         return SEMGET_FAILED;
26     }
27
28     semctl(sid, ACT_RDRS, SETVAL, 0);
29     semctl(sid, ACT_WRTR, SETVAL, 0);
30     semctl(sid, WRITE_QUEUE, SETVAL, 0);
31     semctl(sid, READ_QUEUE, SETVAL, 0);
32
33     int child_pid;
34     for (size_t i = 0; i < READERS_AMOUNT; i++) {
35         switch ((child_pid = fork())) {
36         case -1:
37             perror("Failed to fork for producer!");
38             exit(FORK_FAILED);
39             break;
40         case 0:
```

```
41              run_reader(shared_counter, sid, i);
42              return 0;
43          }
44      }
45
46      for (int i = 0; i < WRITERS_AMOUNT; i++) {
47          switch ((child_pid = fork())) {
48          case -1:
49              perror("Failed␣to␣fork␣for␣consumer!");
50              exit(FORK_FAILED);
51              break;
52          case 0:
53              run_writer(shared_counter, sid, i);
54              return 0;
55          }
56      }
57
58      for (size_t i = 0; i < WRITERS_AMOUNT + READERS_AMOUNT; i++) {
59          int status;
60          if (wait(&status) == -1) {
61              perror("Something␣wrong␣with␣children␣waiting!");
62              exit(WAIT_FAILED);
63          }
64          if (!WIFEXITED(status))
65              printf("One␣of␣children␣terminated␣abnormally\n");
66      }
67
68      if (shmdt((void*)shared_counter) == -1
69              || shmctl(fd, IPC_RMID, NULL) == -1
70              || semctl(sid, IPC_RMID, 0) == -1) {
71          perror("Something␣went␣wrong␣during␣shutdown!");
72          return SHUTDOWN_FAILED;
73      }
74
75      return 0;
76  }
```

Листинг 2.3: Главный файл программы