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.

Москва — 2020 г.

1 | Задача «Производство-потребление»

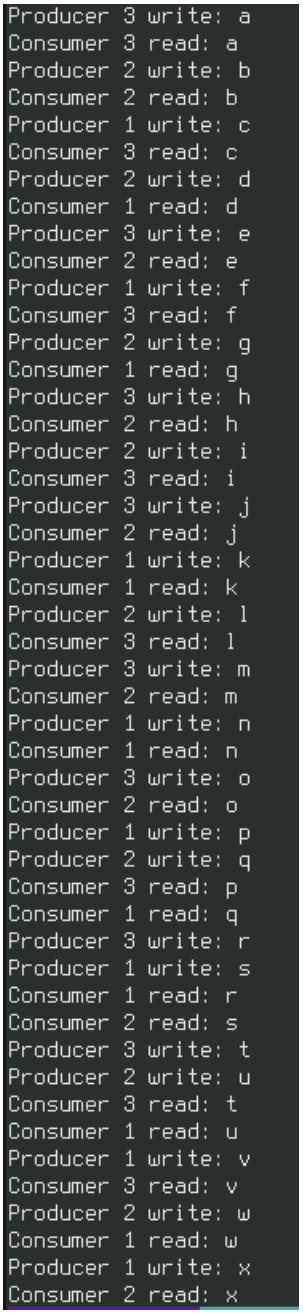

Рис. 1.1: Демонстрация работы программы «Производство-потребление». Задержка потребителя: от 1 до 4, задержка производителя: от 1 до 4

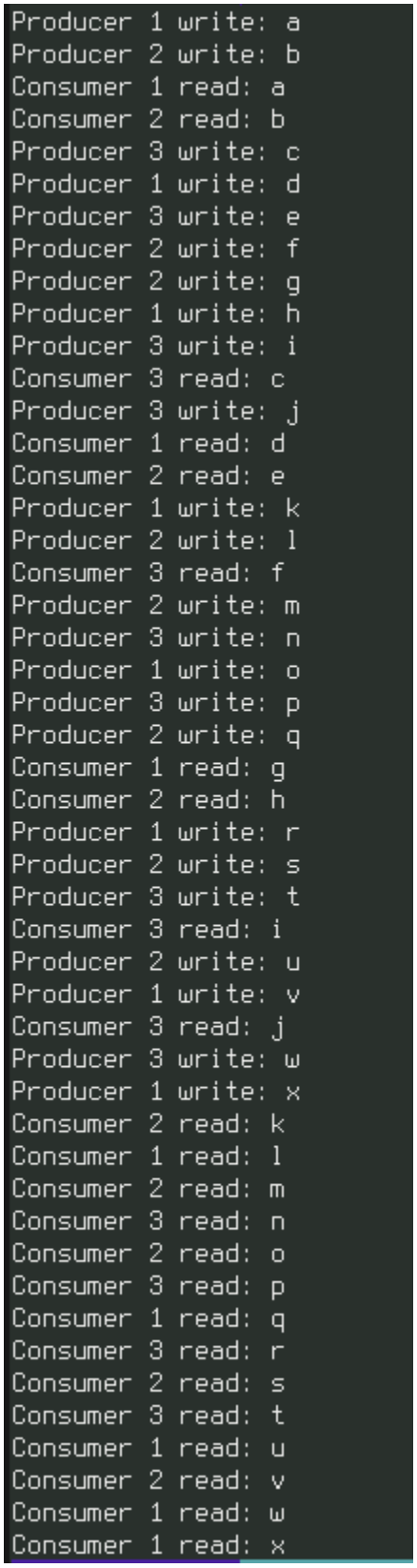

Рис. 1.2: Демонстрация работы программы «Производство-потребление». Задержка потребителя: от 1 до 9, задержка производителя: от 1 до 4

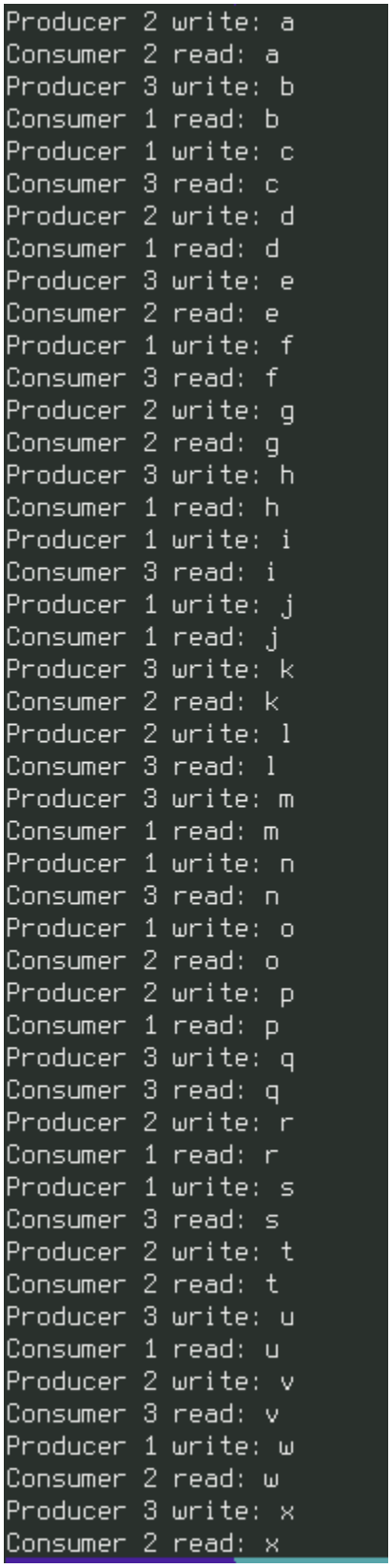

Рис. 1.3: Демонстрация работы программы «Производство-потребление». Задержка потребителя: от 1 до 4, задержка производителя: от 1 до 9

1.1 Листинги кода

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

Листинг 1.1: Реализация очереди на основе циклического буффера

```
#include "buffer.h"

int init_buffer(cbuffer_t *const buf) {
  if (!buf) {
    perror("Error while initializing buffer.\n");
    return BUF_ERR;
  }

  return OK;
}

int write_buffer(cbuffer_t *const buf, const char element) {
  if (!buf) {
    return BUF_ERR;
  }

  buf->buffer[buf->write_pos++] = element;
  buf->write_pos %= N;

  return OK;
}

int read_buffer(cbuffer_t *buf, char *const element) {
  if (!buf) {
    return BUF_ERR;
  }

  if (!element) {
    return BUF_ERR;
  }

  *element = buf->buffer[buf->read_pos++];
  buf->read_pos %= N;

  return OK;
}
```

Листинг 1.2: Реализация «потребителей»

```
#include "consumer.h"

struct sembuf CONS_LOCK[] = {
  { BUFF_FULL, -1, 0 },
  { BIN_SEM, -1, 0 }
```

```
};

struct sembuf CONS_RELEASE[] = {
  { BUFF_EMPTY, 1, 0 },
  { BIN_SEM, 1, 0 }
};

int consumer_run(cbuffer_t *const buf, const int sid, const int consid) {
  srand(time(NULL) + consid);

  if (!buf) {
    return BUF_ERR;
  }

  for (int i = 0; i < ITER_CNT; i++) {
    int sleep_time = rand() % CONS_TIME_RANGE + CONS_TIME_START;
    sleep(sleep_time);

    if (-1 == semop(sid, CONS_LOCK, SEM_SIZE)) {

    return CONS_LOCK_ERROR;
    }

    char symb;

    if (-1 == read_buffer(buf, &symb)) {
      return BUFF_READ_ERROR;
    }

    fprintf(stdout, "Consumer %d read: %c\n", consid + 1, symb);

    if (-1 == semop(sid, CONS_RELEASE, SEM_SIZE)) {
      return CONS_RELEASE_ERROR;
    }
  }

  return OK;
}
```

Листинг 1.3: Реализация «производителей»

```
#include "producer.h"

struct sembuf PROD_LOCK[] = {
  { BUFF_EMPTY, -1, 0 },
  { BIN_SEM, -1, 0 }
};

struct sembuf PROD_RELEASE[] = {
  { 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
        == semctl(sid, IPC_RMID, 0)) {
100     perror("Error while shutdown.\n");
101     return SHUTDOWN_ERROR;
102   }
103
104   return OK;
105 }
```

2 Задача «Читатели-писатели»

2.1 Демонстрация работы программы

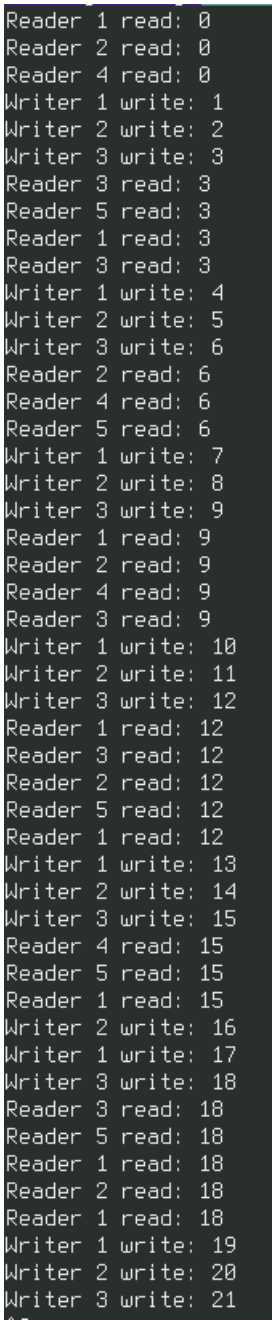

Рис. 2.1: Демонстрация работы программы «Читатели-писатели».

2.2 Листинги кода

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

Листинг 2.1: Реализация «читателей» и «писателей»

```
#include "read_write.h"

struct sembuf READER_QUEUE[] = {
  { READ_QUEUE, 1, 0 },
  { ACTIVE_WRITER, 0, 0 },
  { WRITE_QUEUE, 0, 0 },
};

struct sembuf READER_LOCK[] = {
  { ACTIVE_READER, 1, 0 },
  { READ_QUEUE, -1, 0 },
};

struct sembuf READER_RELEASE[] = {
  { ACTIVE_READER, -1, 0 },
};

struct sembuf WRITER_LOCK[] = {
  { ACTIVE_WRITER, 1, 0 },
  { WRITE_QUEUE, -1, 0 },
};

struct sembuf WRITER_RELEASE[] = {
  { ACTIVE_WRITER, -1, 0 },
};

struct sembuf WRITER_QUEUE[] = {
  { WRITE_QUEUE, 1, 0 },
  { ACTIVE_READER, 0, 0 },
  { ACTIVE_WRITER, 0, 0 },
};

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 start_write(int sid) {
  return semop(sid, WRITER_QUEUE, 3) != -1 && semop(sid, WRITER_LOCK, 2) !=
      -1;
}
```

```
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  }
```