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 Богаченко А.Е.

Группа ИУ7-56Б

Оценка (баллы) _____

Преподаватели Рязанова Н. Ю.

Москва — 2020 г.

Задача «Производство-потребление»

Листинги кода

```c
#include "buffer.h"

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

    return 0;
}

int buff_write(cycle_buff_t *const buffer, const char c) {
    if (!buffer) {
        return -1;
    }
    buffer->data[buffer->wpos++] = c;
    buffer->wpos %= N;

    return 0;
}

int buff_read(cycle_buff_t *const buffer, char *const dst) {
    if (!buffer) {
        return -1;
    }
    *dst = buffer->data[buffer->rpos++];
    buffer->rpos %= N;

    return 0;
}
```

Листинг 1 – Очередь на основе циклического массива (буфера). Код.

```c
#ifndef __BUFFER_H__
#define __BUFFER_H__

#include <string.h>
#include <unistd.h>

#define N 64
```

```
8
9  typedef char data_t[N];
10
11 typedef struct {
12     size_t rpos;
13     size_t wpos;
14     data_t data;
15 } cycle_buff_t;
16
17 int buff_init(cycle_buff_t *const buffer);
18 int buff_write(cycle_buff_t *const buffer, const char c);
19 int buff_read(cycle_buff_t *const buffer, char* const dst);
20
21 #endif  // __BUFFER_H__
```

Листинг 2 – Очередь на основе циклического массива (буфера). Заголовочник.

```
1  #include "runners.h"
2
3  struct sembuf PROD_LOCK[2] = {{BUF_EMPTY, -1, 0}, {BIN_SEM, -1, 0}};
4  struct sembuf PROD_RELEASE[2] = {{BUF_FULL, 1, 0}, {BIN_SEM, 1, 0}};
5
6  struct sembuf CONS_LOCK[2] = {{BUF_FULL, -1, 0}, {BIN_SEM, -1, 0}};
7  struct sembuf CONS_RELEASE[2] = {{BUF_EMPTY, 1, 0}, {BIN_SEM, 1, 0}};
8
9  int run_producer(cycle_buff_t *const buffer,
10                   const int sid,
11                   const int prod_id) {
12     if (!buffer) {
13         return -1;
14     }
15
16     srand(time(NULL) + prod_id);
17
18     int sleep_time;
19     char ch;
20     for (short i = 0; i < ITERATIONS_AMOUNT; ++i) {
21         sleep_time = rand() % MAX_RANDOM_PROD + 1;
22         sleep(sleep_time);
23
24         if (semop(sid, PROD_LOCK, 2) == -1) {
25             perror("prod lock error!");
26             exit(EXIT_FAILURE);
27         }
28
29         // !!! ---- CRITICAL ---- !!!
30         ch = 'a' + (char)(buffer->wpos % 26);
31         if (buff_write(buffer, ch) == -1) {
```

```
32              perror("buffer write error!");
33              return EXIT_FAILURE;
34          }
35          printf(" Producer #%d write: %c — idle %ds\n", prod_id,
36                 ch, sleep_time);
37          // !!! —— CRITICAL —— !!!
38
39          if (semop(sid, PROD_RELEASE, 2) == -1) {
40              perror("prod release error!");
41              exit(EXIT_FAILURE);
42          }
43      }
44      return EXIT_SUCCESS;
45  }
46
47  int run_consumer(cycle_buff_t *const buffer,
48                   const int sid,
49                   const int cons_id) {
50      if (!buffer) {
51          return -1;
52      }
53
54      srand(time(NULL) + cons_id + PROD_COUNT);
55
56      int sleep_time;
57      char ch;
58      for (short i = 0; i < ITERATIONS_AMOUNT; ++i) {
59          sleep_time = rand() % MAX_RANDOM_CONS + 1;
60          sleep(sleep_time);
61
62          if (semop(sid, CONS_LOCK, 2) == -1) {
63              perror("consumer lock error!");
64              exit(EXIT_FAILURE);
65          }
66
67          // !!! —— CRITICAL —— !!!
68          if (buff_read(buffer, &ch) == -1) {
69              perror("buffer read error!");
70              return EXIT_FAILURE;
71          }
72          printf(" Consumer #%d read:  %c — idle %ds\n", cons_id,
73                 ch, sleep_time);
74          // !!! —— CRITICAL —— !!!
75
76          if (semop(sid, CONS_RELEASE, 2) == -1) {
77              perror("consumer release error!");
78              exit(EXIT_FAILURE);
79          }
```

```
80      }
81      return EXIT_SUCCESS;
82  }
```

Листинг 3 – Реализация задачи. Код.

```
1  #ifndef __RUNNERS_H__
2  #define __RUNNERS_H__
3
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <sys/sem.h>
7  #include <time.h>
8
9  #include "buffer.h"
10
11 #define ITERATIONS_AMOUNT 8
12
13 #define CONS_COUNT 3
14 #define PROD_COUNT 3
15
16 #define BIN_SEM 0
17 #define BUF_FULL 1
18 #define BUF_EMPTY 2
19
20 #define MAX_RANDOM_PROD 2
21 #define MAX_RANDOM_CONS 5
22
23 int run_producer(cycle_buff_t *const buffer, const int sid, const int prod_id)
       ;
24 int run_consumer(cycle_buff_t *const buffer, const int sid, const int cons_id)
       ;
25
26 #endif  // __RUNNERS_H__
```

Листинг 4 – Реализация задачи. Заголовочник.

```
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 <time.h>
10 #include <unistd.h>
11 #include <wait.h>
12
```

```c
#include "buffer.h"
#include "runners.h"

#define MAX_SEMS 3

int main(void) {
    setbuf(stdout, NULL);
    int fd = shmget(IPC_PRIVATE,
                    sizeof(cycle_buff_t),
                    IPC_CREAT | S_IRWXU | S_IRWXG | S_IRWXO);
    if (fd == -1) {
        perror("shmget failed!");
        return EXIT_FAILURE;
    }

    cycle_buff_t *buffer;
    if ((buffer = shmat(fd, 0, 0)) == (void *)-1) {
        perror("shmat failed!");
        return EXIT_FAILURE;
    }

    if (buff_init(buffer) == -1) {
        perror("init failed!");
        return EXIT_FAILURE;
    }

    int sid = semget(IPC_PRIVATE,
                     MAX_SEMS,
                     IPC_CREAT | S_IRWXU | S_IRWXG | S_IRWXO);
    if (sid == -1) {
        perror("semget failed!");
        return EXIT_FAILURE;
    }

    semctl(sid, BIN_SEM, SETVAL, 1);
    semctl(sid, BUF_EMPTY, SETVAL, N);
    semctl(sid, BUF_FULL, SETVAL, 0);

    int child_pid;
    for (short i = 0; i < PROD_COUNT; ++i) {
        switch ((child_pid = fork())) {
            case -1:
                perror("producer fork failed!");
                exit(EXIT_FAILURE);
                break;
            case 0:
                run_producer(buffer, sid, i);
                return EXIT_SUCCESS;
```

```
61          }
62      }
63
64      for (short i = 0; i < CONS_COUNT; ++i) {
65          switch ((child_pid = fork())) {
66              case -1:
67                  perror("consumer fork failed!");
68                  exit(EXIT_FAILURE);
69                  break;
70              case 0:
71                  run_consumer(buffer, sid, i);
72                  return EXIT_SUCCESS;
73          }
74      }
75
76      for (short i = 0; i < CONS_COUNT + PROD_COUNT; ++i) {
77          int status;
78          if (wait(&status) == -1) {
79              perror("children error!");
80              exit(EXIT_FAILURE);
81          }
82          if (!WIFEXITED(status))
83              puts("unexpected termination");
84      }
85
86      if (shmdt((void *)buffer) == -1 ||
87          shmctl(fd, IPC_RMID, NULL) == -1 ||
88          semctl(sid, IPC_RMID, 0) == -1) {
89          perror("exit error!");
90
91          return EXIT_FAILURE;
92      }
93
94      return EXIT_SUCCESS;
95  }
```

Листинг 5 – Точка входа в программу

Работа программы

```
┌──(root💀NebuchadnezzaR)-[~/bmstu-os/lab5/pc]
└─# ./app.exe
Producer #0 write: a -- idle 1s
Producer #1 write: b -- idle 1s
Producer #2 write: c -- idle 1s
Producer #0 write: d -- idle 1s
Consumer #1 read:  a -- idle 2s
Producer #1 write: e -- idle 1s
Producer #2 write: f -- idle 1s
Producer #0 write: g -- idle 1s
Producer #2 write: h -- idle 1s
Consumer #0 read:  b -- idle 4s
Consumer #2 read:  c -- idle 4s
Consumer #1 read:  d -- idle 2s
Producer #1 write: i -- idle 2s
Producer #0 write: j -- idle 1s
Producer #2 write: k -- idle 2s
Producer #0 write: l -- idle 2s
Producer #1 write: m -- idle 2s
Producer #2 write: n -- idle 1s
Producer #0 write: o -- idle 2s
Producer #1 write: p -- idle 2s
Producer #2 write: q -- idle 2s
Consumer #0 read:  e -- idle 5s
Consumer #2 read:  f -- idle 5s
Consumer #1 read:  g -- idle 5s
Producer #1 write: r -- idle 1s
Producer #2 write: s -- idle 2s
Producer #0 write: t -- idle 2s
Consumer #2 read:  h -- idle 2s
Producer #1 write: u -- idle 2s
Producer #0 write: v -- idle 1s
Producer #1 write: w -- idle 1s
Producer #2 write: x -- idle 2s
Consumer #2 read:  i -- idle 2s
Consumer #0 read:  j -- idle 5s
Consumer #1 read:  k -- idle 5s
```

Рисунок 1 – «Производство-Потребление». Максимальная задержка потребителя – 5с, производителя – 2с.

Задача «Читатели-Писатели»

Листинги кода

```c
#include "io.h"

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

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

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

struct sembuf WRITER_QUEUE[] = {
    {WRITE_QUEUE, 1, 0},
    {READER, 0, 0},
    {WRITER, 0, 0},
};

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

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

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

int reader_run(int *const shared_counter,
               const int sid,
```

```
43                 const int reader_id) {
44     if (!shared_counter) {
45         return -1;
46     }
47
48     srand(time(NULL) + reader_id);
49
50     int sleep_time;
51     for (short i = 0; i < ITERATIONS; ++i) {
52         sleep_time = rand() % MAX_RANDOM + 1;
53         sleep(sleep_time);
54
55         if (!start_read(sid)) {
56             perror("Something went wrong with start_read!");
57             exit(EXIT_FAILURE);
58         }
59
60         // !!! ---- CRITICAL ---- !!!
61         int val = *shared_counter;
62         printf(" Reader #%d read:  %3d --- idle %ds\n", reader_id,
63                 val, sleep_time);
64         // !!! ---- CRITICAL ---- !!!
65
66         if (!stop_read(sid)) {
67             perror("Something went wrong with stop_read!");
68             exit(EXIT_FAILURE);
69         }
70     }
71     return EXIT_SUCCESS;
72 }
73
74 static inline int write_start(int sid) {
75     return semop(sid, WRITER_QUEUE, 3) != -1 &&
76            semop(sid, WRITER_LOCK, 2) != -1;
77 }
78
79 static inline int write_stop(int sid) {
80     return semop(sid, WRITER_RELEASE, 1) != -1;
81 }
82
83 int writer_run(int *const shared_counter,
84                 const int sid,
85                 const int writer_id) {
86     if (!shared_counter) {
87         return -1;
88     }
89
90     srand(time(NULL) + writer_id + READERS_COUNT);
```

```
91
92      int sleep_time;
93      for (short i = 0; i < ITERATIONS; ++i) {
94          sleep_time = rand() % MAX_RANDOM + 1;
95          sleep(sleep_time);
96
97          if (!write_start(sid)) {
98              perror("Something went wrong with write_start!");
99              exit(EXIT_FAILURE);
100         }
101
102         // !!! ---- CRITICAL ---- !!!
103         int val = ++(*shared_counter);
104         printf(" Writer #%d write: %3d --- idle %ds\n", writer_id,
105                 val, sleep_time);
106         // !!! ---- CRITICAL ---- !!!
107
108         if (!write_stop(sid)) {
109             perror("Something went wrong with write_stop!");
110             exit(EXIT_FAILURE);
111         }
112     }
113
114     return EXIT_SUCCESS;
115 }
```

Листинг 6 – Реализация задачи. Код.

```
1  #ifndef __IO_H__
2  #define __IO_H__
3
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <sys/sem.h>
7  #include <time.h>
8  #include <unistd.h>
9
10 #define ITERATIONS 20
11 #define WRITERS_COUNT 3
12 #define READERS_COUNT 5
13
14 #define MAX_SEMS 4
15 #define READER 0
16 #define WRITER 1
17
18 #define READ_QUEUE 2
19 #define WRITE_QUEUE 3
20
21 #define MAX_RANDOM 3
```

```
22
23 int reader_run(int *const shared_counter, const int sid, const int reader_id);
24 int writer_run(int *const shared_counter, const int sid, const int writer_id);
25
26 #endif // __IO_H__
```

Листинг 7 – Реализация задачи. Заголовочник.

```
1  #include <sys/shm.h>
2  #include <sys/stat.h>
3  #include <wait.h>
4
5  #include "io.h"
6
7  int main(void) {
8      setbuf(stdout, NULL);
9      int fd = shmget(IPC_PRIVATE,
10                     sizeof(int),
11                     IPC_CREAT | S_IRWXU | S_IRWXG | S_IRWXO);
12     if (fd == -1) {
13         perror("shmget failed!");
14         return EXIT_FAILURE;
15     }
16
17     int *shared_counter;
18     if ((shared_counter = shmat(fd, 0, 0)) == (void *)-1) {
19         perror("shmat failed!");
20         return EXIT_FAILURE;
21     }
22
23     int sid = semget(IPC_PRIVATE,
24                      MAX_SEMS,
25                      IPC_CREAT | S_IRWXU | S_IRWXG | S_IRWXO);
26     if (sid == -1) {
27         perror("semget failed!");
28         return EXIT_FAILURE;
29     }
30
31     semctl(sid, READER, SETVAL, 0);
32     semctl(sid, WRITER, SETVAL, 0);
33     semctl(sid, WRITE_QUEUE, SETVAL, 0);
34     semctl(sid, READ_QUEUE, SETVAL, 0);
35
36     int child_pid;
37     for (short i = 0; i < READERS_COUNT; ++i) {
38         switch ((child_pid = fork())) {
39             case -1:
40                 perror("reader fork failed!");
41                 exit(EXIT_FAILURE);
```

```
42                  break;
43              case 0:
44                  reader_run(shared_counter, sid, i);
45                  return EXIT_SUCCESS;
46          }
47      }
48
49      for (short i = 0; i < WRITERS_COUNT; ++i) {
50          switch ((child_pid = fork())) {
51              case -1:
52                  perror("writer fork failed!");
53                  exit(EXIT_FAILURE);
54                  break;
55              case 0:
56                  writer_run(shared_counter, sid, i);
57                  return EXIT_SUCCESS;
58          }
59      }
60
61      for (short i = 0; i < WRITERS_COUNT + READERS_COUNT; ++i) {
62          int status;
63          if (wait(&status) == -1) {
64              perror("children error!");
65              exit(EXIT_FAILURE);
66          }
67
68          if (!WIFEXITED(status)) {
69              puts("unexpected termination");
70          }
71      }
72
73      if (shmdt((void *)shared_counter) == -1 ||
74          shmctl(fd, IPC_RMID, NULL) == -1 ||
75          semctl(sid, IPC_RMID, 0) == -1) {
76
77          perror("exit error!");
78
79          return EXIT_FAILURE;
80      }
81
82      return EXIT_SUCCESS;
83  }
```

Листинг 8 – Точка входа в программу

Работа программы

```
┌──(root💀NebuchadnezzaR)-[~/bmstu-os/lab5/rw]
└─# ./app.exe
Reader #0 read:    0 -- idle 1s
Reader #3 read:    0 -- idle 1s
Reader #1 read:    0 -- idle 2s
Reader #2 read:    0 -- idle 2s
Reader #4 read:    0 -- idle 2s
Writer #0 write:   1 -- idle 2s
Writer #2 write:   2 -- idle 2s
Reader #0 read:    2 -- idle 2s
Reader #3 read:    2 -- idle 2s
Reader #4 read:    2 -- idle 1s
Writer #1 write:   3 -- idle 3s
Writer #0 write:   4 -- idle 1s
Writer #2 write:   5 -- idle 1s
Reader #0 read:    5 -- idle 1s
Reader #3 read:    5 -- idle 1s
Writer #1 write:   6 -- idle 1s
Writer #0 write:   7 -- idle 1s
Reader #2 read:    7 -- idle 3s
Reader #1 read:    7 -- idle 3s
Reader #0 read:    7 -- idle 1s
Reader #3 read:    7 -- idle 1s
Writer #1 write:   8 -- idle 1s
Reader #4 read:    8 -- idle 3s
Writer #2 write:   9 -- idle 3s
Writer #0 write:  10 -- idle 2s
Writer #1 write:  11 -- idle 1s
Reader #2 read:   11 -- idle 2s
Reader #1 read:   11 -- idle 2s
Reader #3 read:   11 -- idle 2s
Writer #2 write:  12 -- idle 1s
Reader #0 read:   12 -- idle 3s
Writer #0 write:  13 -- idle 2s
Reader #4 read:   13 -- idle 2s
Writer #1 write:  14 -- idle 3s
Writer #2 write:  15 -- idle 2s
Writer #0 write:  16 -- idle 1s
```

Рисунок 2 – «Читатели-Писатели». Максимальная задержка – 3с.