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 Кононенко С.С.

Группа ИУ7-53Б

Оценка (баллы) _____

Преподаватели Рязанова Н.Ю.

Москва — 2020 г.

Задача «Производство – потребление»

Листинг 1 – Реализация задачи «производство – потребление»

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <time.h>
#include <wait.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <sys/types.h>

#define N 24
#define MAX_SEMS 3
#define ITERS 8

#define P_COUNT 3
#define C_COUNT 3

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

#define MAX_RAND_P 2
#define MAX_RAND_C 5

typedef char data_t[N];
typedef struct
{
    size_t r_pos;
    size_t w_pos;
    data_t data;
} cbuf_t;

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

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

int init_buf(cbuf_t *const buf)
```

```
42 {
43     if (!buf)
44     {
45         return -1;
46     }
47     memset(buf, 0, sizeof(cbuf_t));
48 
49     return 0;
50 }
51 
52 int write_buf(cbuf_t *const buf, const char c)
53 {
54     if (!buf)
55     {
56         return -1;
57     }
58     buf->data[buf->w_pos++] = c;
59     buf->w_pos %= N;
60 
61     return 0;
62 }
63 
64 int read_buf(cbuf_t *const buf, char *const dst)
65 {
66     if (!buf)
67     {
68         return -1;
69     }
70     *dst = buf->data[buf->r_pos++];
71     buf->r_pos %= N;
72 
73     return 0;
74 }
75 
76 int p_run(cbuf_t *const buf, const int s_id, const int p_id)
77 {
78     if (!buf)
79     {
80         return -1;
81     }
82 
83     srand(time(NULL) + p_id);
84 
85     int stime;
86     char ch;
87 
88     for (size_t i = 0; i < ITERS; ++i)
89     {
```

```
90          stime = rand() % MAX_RAND_P + 1;
91          sleep(stime);
92
93          if (semop(s_id, P_LOCK, 2) == -1)
94          {
95              perror("Producer lock error.");
96
97              exit(EXIT_FAILURE);
98          }
99
100         ch = 'a' + (char)(buf->w_pos % 26);
101
102         if (write_buf(buf, ch) == -1)
103         {
104             perror("Buffer write error.");
105
106             return EXIT_FAILURE;
107         }
108         printf("!Producer #%d wrote: %c // Idle time: %ds\n", p_id, ch,
                stime);
109
110         if (semop(s_id, P_RELEASE, 2) == -1)
111         {
112             perror("Producer release error.");
113
114             exit(EXIT_FAILURE);
115         }
116     }
117
118     return EXIT_SUCCESS;
119 }
120
121 int c_run(cbuf_t *const buf, const int s_id, const int c_id)
122 {
123     if (!buf)
124     {
125         return -1;
126     }
127
128     srand(time(NULL) + c_id + P_COUNT);
129
130     int stime;
131     char ch;
132
133     for (size_t i = 0; i < ITERS; ++i)
134     {
135         stime = rand() % MAX_RAND_C + 1;
136         sleep(stime);
```

```
137
138         if (semop(s_id, C_LOCK, 2) == -1)
139         {
140             perror("Consumer lock error.");
141
142             exit(EXIT_FAILURE);
143         }
144
145         if (read_buf(buf, &ch) == -1)
146         {
147             perror("Buffer read error.");
148
149             return EXIT_FAILURE;
150         }
151         printf("?Consumer #%d read:  %c // Idle time: %ds\n", c_id, ch,
                stime);
152
153         if (semop(s_id, C_RELEASE, 2) == -1)
154         {
155             perror("Consumer release error.");
156
157             exit(EXIT_FAILURE);
158         }
159     }
160
161     return EXIT_SUCCESS;
162 }
163
164 int main()
165 {
166     setbuf(stdout, NULL);
167
168     int fd = shmget(IPC_PRIVATE, sizeof(cbuf_t), IPC_CREAT | S_IRWXU |
            S_IRWXG | S_IRWXO);
169     if (fd == -1)
170     {
171         perror("shmget failed.");
172
173         return EXIT_FAILURE;
174     }
175
176     cbuf_t *buf = shmat(fd, 0, 0);
177     if (buf == (void *)-1)
178     {
179         perror("shmat failed.");
180
181         return EXIT_FAILURE;
182     }
```

```
183
184     if (init_buf(buf) == -1)
185     {
186         perror("Buffer␣initialization␣failed.");
187
188         return EXIT_FAILURE;
189     }
190
191     int s_id = semget(IPC_PRIVATE, MAX_SEMS, IPC_CREAT | S_IRWXU | S_IRWXG
            | S_IRWXO);
192     if (s_id == -1)
193     {
194         perror("semget␣failed.");
195
196         return EXIT_FAILURE;
197     }
198
199     semctl(s_id, BIN_SEM, SETVAL, 1);
200     semctl(s_id, BUF_EMPTY, SETVAL, N);
201     semctl(s_id, BUF_FULL, SETVAL, 0);
202
203     int chpid;
204     for (size_t i = 0; i < P_COUNT; ++i)
205     {
206         switch ((chpid = fork()))
207         {
208         case -1:
209             perror("Producer␣fork␣failed.");
210
211             exit(EXIT_FAILURE);
212             break;
213         case 0:
214             p_run(buf, s_id, i);
215
216             return EXIT_SUCCESS;
217         }
218     }
219
220     for (size_t i = 0; i < C_COUNT; ++i)
221     {
222         switch ((chpid = fork()))
223         {
224         case -1:
225             perror("Consumer␣fork␣failed.");
226
227             exit(EXIT_FAILURE);
228             break;
229         case 0:
```

```
230             c_run(buf, s_id, i);
231             return EXIT_SUCCESS;
232         }
233     }
234
235     for (size_t i = 0; i < C_COUNT + P_COUNT; ++i)
236     {
237         int status;
238         if (wait(&status) == -1)
239         {
240             perror("Child error.");
241
242             exit(EXIT_FAILURE);
243         }
244         if (!WIFEXITED(status))
245         {
246             printf("Child process terminated abnormally\n");
247         }
248     }
249
250     if (shmdt((void *)buf) == -1 ||
251         shmctl(fd, IPC_RMID, NULL) == -1 ||
252         semctl(s_id, IPC_RMID, 0) == -1)
253     {
254         perror("Exit error.");
255
256         return EXIT_FAILURE;
257     }
258
259     return EXIT_SUCCESS;
260 }
```

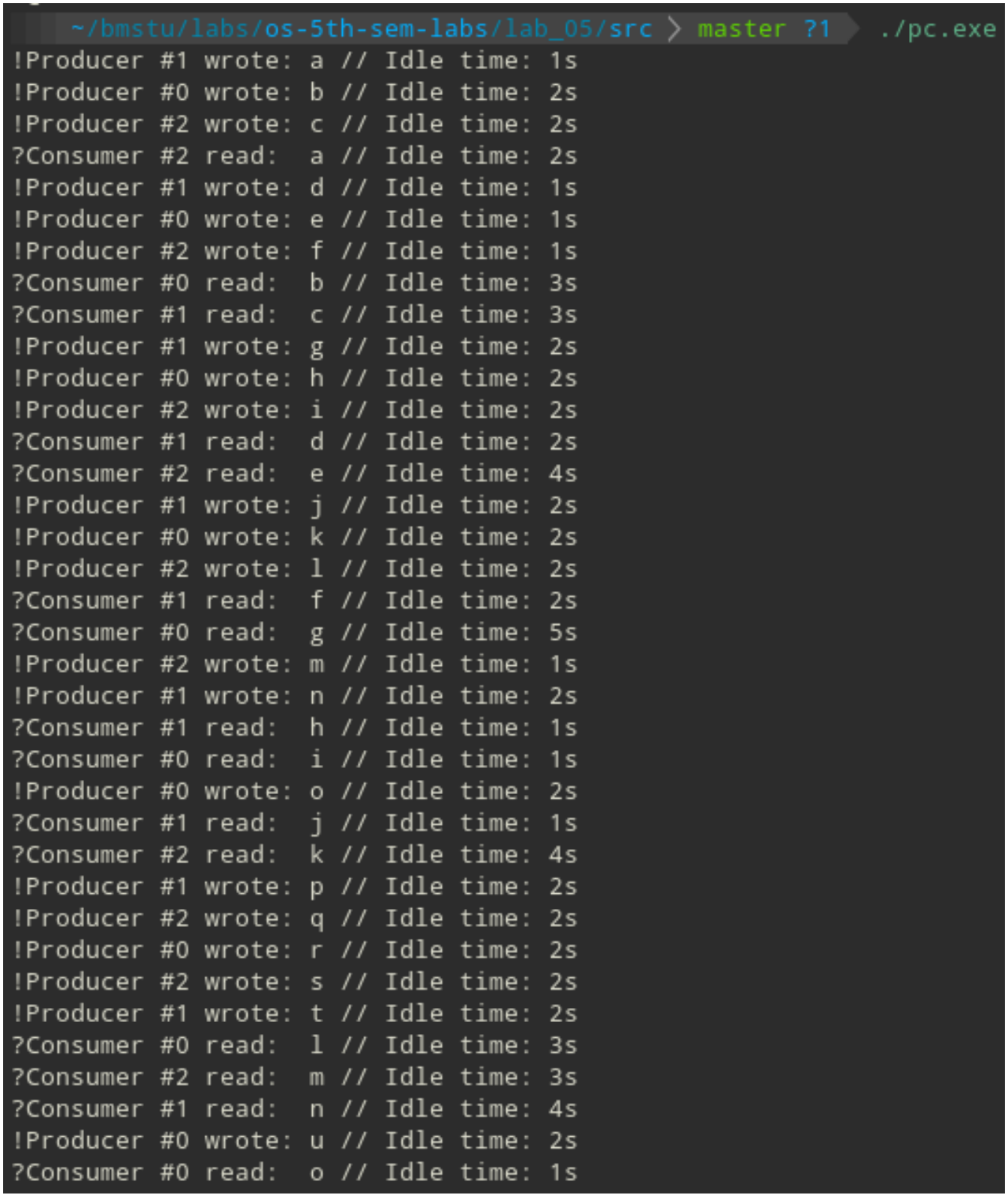

Рисунок 1 – Демонстрация работы программы. Максимальная задержка потребителя – 5 секунд, производителя – 2 секунды

Задача «Читатели – писатели»

Листинг 2 – Реализация задачи «читатели – писатели»

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>
#include <wait.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <sys/stat.h>

#define MAX_SEMS 4
#define ITERS 20

#define READERS_COUNT 5
#define WRITERS_COUNT 3

#define ACTIVE_READERS 0
#define ACTIVE_WRITERS 1

#define WAITING_READERS 2
#define WAITING_WRITERS 3

#define MAX_RAND 3

struct sembuf START_READ[] = {
    {WAITING_READERS, 1, 0},
    {ACTIVE_WRITERS, 0, 0},
    {WAITING_WRITERS, 0, 0},
    {ACTIVE_READERS, 1, 0},
    {WAITING_READERS, -1, 0},
};

struct sembuf STOP_READ[] = {
    {ACTIVE_READERS, -1, 0},
};

struct sembuf START_WRITE[] = {
    {WAITING_WRITERS, 1, 0},
    {ACTIVE_READERS, 0, 0},
    {ACTIVE_WRITERS, 0, 0},
    {ACTIVE_WRITERS, 1, 0},
    {WAITING_WRITERS, -1, 0},
};

```

```
44 struct sembuf STOP_WRITE[] = {
45     {ACTIVE_WRITERS, -1, 0},
46 };
47
48 int start_read(int s_id)
49 {
50     return semop(s_id, START_READ, 5) != -1;
51 }
52
53 int stop_read(int s_id)
54 {
55     return semop(s_id, STOP_READ, 1) != -1;
56 }
57
58 int start_write(int s_id)
59 {
60     return semop(s_id, START_WRITE, 5) != -1;
61 }
62
63 int stop_write(int s_id)
64 {
65     return semop(s_id, STOP_WRITE, 1) != -1;
66 }
67
68 int rr_run(int *const shcntr, const int s_id, const int r_id)
69 {
70     if (!shcntr)
71     {
72         return -1;
73     }
74
75     srand(time(NULL) + r_id);
76
77     int stime;
78
79     for (size_t i = 0; i < ITERS; ++i)
80     {
81         stime = rand() % MAX_RAND + 1;
82         sleep(stime);
83
84         if (!start_read(s_id))
85         {
86             perror("Reading␣start␣error.");
87
88             exit(EXIT_FAILURE);
89         }
90
91         int val = *shcntr;
```

```
92          printf("?Reader #%d read:  %3d // Idle time: %ds\n", r_id, val,
                stime);
93
94          if (!stop_read(s_id))
95          {
96              perror("Reading end error.");
97
98              exit(EXIT_FAILURE);
99          }
100     }
101
102     return EXIT_SUCCESS;
103 }
104
105 int wr_run(int *const shcntr, const int s_id, const int w_id)
106 {
107     if (!shcntr)
108     {
109         return -1;
110     }
111
112     srand(time(NULL) + w_id + READERS_COUNT);
113
114     int stime;
115
116     for (size_t i = 0; i < ITERS; ++i)
117     {
118         stime = rand() % MAX_RAND + 1;
119         sleep(stime);
120
121         if (!start_write(s_id))
122         {
123             perror("Writing start error.");
124
125             exit(EXIT_FAILURE);
126         }
127
128         int val = ++(*shcntr);
129         printf("!Writer #%d wrote:  %3d // Idle time: %ds\n", w_id, val,
                stime);
130
131         if (!stop_write(s_id))
132         {
133             perror("Writing end error.");
134
135             exit(EXIT_FAILURE);
136         }
137     }
```

```
138
139     return EXIT_SUCCESS;
140 }
141
142 int main()
143 {
144     setbuf(stdout, NULL);
145
146     int fd = shmget(IPC_PRIVATE, sizeof(int), IPC_CREAT | S_IRWXU |
            S_IRWXG | S_IRWXO);
147     if (fd == -1)
148     {
149         perror("shmget failed.");
150
151         return EXIT_FAILURE;
152     }
153
154     int *shcntr = shmat(fd, 0, 0);
155     if (shcntr == (void *)-1)
156     {
157         perror("shmat failed.");
158
159         return EXIT_FAILURE;
160     }
161
162     int s_id = semget(IPC_PRIVATE, MAX_SEMS, IPC_CREAT | S_IRWXU | S_IRWXG
            | S_IRWXO);
163     if (s_id == -1)
164     {
165         perror("semget failed.");
166
167         return EXIT_FAILURE;
168     }
169
170     semctl(s_id, ACTIVE_READERS, SETVAL, 0);
171     semctl(s_id, ACTIVE_WRITERS, SETVAL, 0);
172     semctl(s_id, WAITING_WRITERS, SETVAL, 0);
173     semctl(s_id, WAITING_READERS, SETVAL, 0);
174
175     int chpid;
176     for (size_t i = 0; i < READERS_COUNT; ++i)
177     {
178         switch ((chpid = fork()))
179         {
180         case -1:
181             perror("Reader fork failed.");
182
183             exit(EXIT_FAILURE);
```

```
184             break;
185         case 0:
186             rr_run(shcntr, s_id, i);
187 
188             return EXIT_SUCCESS;
189         }
190     }
191 
192     for (size_t i = 0; i < WRITERS_COUNT; ++i)
193     {
194         switch ((chpid = fork()))
195         {
196         case -1:
197             perror("Writer fork failed.");
198 
199             exit(EXIT_FAILURE);
200             break;
201         case 0:
202             wr_run(shcntr, s_id, i);
203 
204             return EXIT_SUCCESS;
205         }
206     }
207 
208     for (size_t i = 0; i < WRITERS_COUNT + READERS_COUNT; ++i)
209     {
210         int status;
211         if (wait(&status) == -1)
212         {
213             perror("Child error.");
214 
215             exit(EXIT_FAILURE);
216         }
217         if (!WIFEXITED(status))
218         {
219             printf("Child process terminated abnormally\n");
220         }
221     }
222 
223     if (shmdt((void *)shcntr) == -1 ||
224         shmctl(fd, IPC_RMID, NULL) == -1 ||
225         semctl(s_id, IPC_RMID, 0) == -1)
226     {
227 
228         perror("Exit error.");
229 
230         return EXIT_FAILURE;
231     }
```

```
232
233     return EXIT_SUCCESS;
234 }
```

```
~/bmstu/labs/os-5th-sem-labs/lab_05/src > master ?1 ./rw.exe
?Reader #2 read:     0 // Idle time: 1s
?Reader #4 read:     0 // Idle time: 1s
!Writer #0 wrote:    1 // Idle time: 1s
!Writer #2 wrote:    2 // Idle time: 1s
?Reader #1 read:     2 // Idle time: 2s
?Reader #3 read:     2 // Idle time: 2s
!Writer #0 wrote:    3 // Idle time: 1s
?Reader #0 read:     3 // Idle time: 3s
!Writer #1 wrote:    4 // Idle time: 3s
?Reader #4 read:     4 // Idle time: 2s
?Reader #0 read:     4 // Idle time: 1s
?Reader #1 read:     4 // Idle time: 2s
?Reader #3 read:     4 // Idle time: 2s
?Reader #2 read:     4 // Idle time: 3s
!Writer #2 wrote:    5 // Idle time: 3s
?Reader #0 read:     5 // Idle time: 1s
?Reader #2 read:     5 // Idle time: 1s
!Writer #1 wrote:    6 // Idle time: 2s
!Writer #0 wrote:    7 // Idle time: 3s
?Reader #3 read:     7 // Idle time: 2s
?Reader #4 read:     7 // Idle time: 3s
!Writer #2 wrote:    8 // Idle time: 2s
?Reader #1 read:     8 // Idle time: 3s
?Reader #0 read:     8 // Idle time: 2s
!Writer #0 wrote:    9 // Idle time: 2s
?Reader #4 read:     9 // Idle time: 1s
?Reader #2 read:     9 // Idle time: 3s
?Reader #1 read:     9 // Idle time: 1s
?Reader #3 read:     9 // Idle time: 2s
!Writer #1 wrote:   10 // Idle time: 3s
!Writer #2 wrote:   11 // Idle time: 2s
?Reader #2 read:    11 // Idle time: 1s
?Reader #0 read:    11 // Idle time: 2s
!Writer #0 wrote:   12 // Idle time: 2s
!Writer #2 wrote:   13 // Idle time: 1s
!Writer #1 wrote:   14 // Idle time: 2s
?Reader #4 read:    14 // Idle time: 3s
```

Рисунок 2 – Демонстрация работы программы. Максимальная задержка – 3 секунды