



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»

# Отчёт

## по лабораторной работе № 5

**Название:** Взаимодействие параллельных процессов

**Дисциплина:** Операционные системы

Студент

ИУ7-55Б

(Группа)

Д.О. Склифасовский

(Подпись, дата)

(И.О. Фамилия)

Преподаватель

Н.Ю. Рязанова

(Подпись, дата)

(И.О. Фамилия)

*Москва, 2020*

## Содержание

<b>1</b>	<b>Задание</b>	<b>3</b>
<b>2</b>	<b>Задание</b>	<b>9</b>

## 1 Задание

Написать программу, реализующую задачу «Производство-потребление» по алгоритму Э. Дейкстры с тремя семафорами: двумя считающими и одним бинарным. В программе должно создаваться не менее 3х процессов - производителей и 3х процессов – потребителей. В программе надо обеспечить случайные задержки выполнения созданных процессов. В программе для взаимодействия производителей и потребителей буфер создается в разделяемом сегменте. Обратите внимание на то, чтобы не работать с одиночной переменной, а работать именно с буфером, состоящим из N ячеек по алгоритму. Производители в ячейки буфера записывают буквы алфавита по порядку. Потребители считывают символы из доступной ячейки. После считывания буквы из ячейки следующий потребитель может взять букву из следующей ячейки.

### Программа:

#### Листинг 1 – Задание 1

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <sys/types.h>
5 #include <sys/ipc.h>
6 #include <sys/sem.h>
7 #include <sys/shm.h>
8 #include <sys/wait.h>
9 #include <sys/stat.h>
10 #include <time.h>
11
12 #define SIZE 24
13
14 #define FULL 0
15 #define EMPTY 1
16 #define BIN 2
```

```

17
18 #define PROD 3
19 #define CONS 3
20
21 char* shared_buffer = NULL;
22 char* cons_pos = 0;
23 char* prod_pos = 0;
24
25 char alph[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
26 int len = 26;
27
28 struct sembuf prod_start[2] = {
29     {EMPTY, -1, 0},
30     {BIN, -1, 0}
31 };
32 struct sembuf prod_stop[2] = {
33     {BIN, 1, 0},
34     {FULL, 1, 0}
35 };
36
37 struct sembuf cons_start[2] = {
38     {FULL, -1, 0},
39     {BIN, -1, 0}
40 };
41 struct sembuf cons_stop[2] = {
42     {BIN, 1, 0},
43     {EMPTY, 1, 0}
44 };
45
46 int perms = S_IRWXU | S_IRWXG | S_IRWXO;
47
48 int get_sem()
49 {
50     int sem = semget(IPC_PRIVATE, 3, IPC_CREAT | perms);
51     if (sem == -1)

```

```

52     {
53         perror("sem\n");
54         exit(1);
55     }
56     int s1 = semctl(sem, FULL, SETVAL, 0);
57     int s2 = semctl(sem, EMPTY, SETVAL, SIZE);
58     int s3 = semctl(sem, BIN, SETVAL, 1);
59
60     return sem;
61 }
62
63 int get_shared_memory()
64 {
65     int shared_memory = shmget(IPC_PRIVATE, (SIZE + 2) * sizeof(
        char), IPC_CREAT | perms);
66     if (shared_memory == -1)
67     {
68         perror("shmget\n");
69         exit(1);
70     }
71
72     prod_pos = (char*)shmat(shared_memory, 0, 0);
73     if (prod_pos == (char*)-1)
74     {
75         perror("shmat\n");
76         exit(1);
77     }
78     cons_pos = prod_pos + sizeof(char);
79     shared_buffer = cons_pos + sizeof(char);
80
81     return shared_memory;
82 }
83
84 void producer(int sem, int id)
85 {

```

```

86     while(1)
87     {
88         if (semop(sem, prod_start, 2) == -1)
89         {
90             perror("semop\n");
91             exit(1);
92         }
93
94         shared_buffer[*prod_pos] = alph[*prod_pos];
95         printf("Producer %d: %c\n", id, alph[*prod_pos]);
96
97         if (++(*prod_pos) == len)
98             (*prod_pos) = 0;
99
100        if (semop(sem, prod_stop, 2) == -1)
101        {
102            perror("semop\n");
103            exit(1);
104        }
105
106        sleep(rand() % 5);
107    }
108 }
109
110 void consumer(int sem, int id)
111 {
112     while(1)
113     {
114         if (semop(sem, cons_start, 2) == -1)
115         {
116             perror("semop\n");
117             exit(1);
118         }
119
120         printf("Consumer %d: %c\n", id, shared_buffer[*cons_pos])

```

```

121         ;
122     if (++(*cons_pos) == len)
123         (*cons_pos) = 0;
124
125     if (semop(sem, cons_stop, 2) == -1)
126     {
127         perror("semop\n");
128         exit(1);
129     }
130
131     sleep(rand() % 5);
132 }
133 }
134
135 void fork_proc(void (*func)(int sem, int id), int sem, int count)
136 {
137     for (int i = 0; i < count; i++)
138     {
139         pid_t pid = fork();
140
141         if (pid == -1)
142         {
143             perror("fork");
144             exit(1);
145         }
146
147         if (pid == 0)
148         {
149             func(sem, i + 1);
150             exit(0);
151         }
152     }
153 }
154

```

```

155 void catch_signal(int signalNum)
156 {
157     printf("Caught\n");
158 }
159
160 int main()
161 {
162     int shared_memory = get_shared_memory();
163     int sem = get_sem();
164
165     fork_proc(producer, sem, PROD);
166     fork_proc(consumer, sem, CONS);
167
168     signal(SIGINT, catch_signal);
169
170     for (int i = 0; i < PROD + CONS; i++)
171     {
172         int status;
173         wait(&status);
174     }
175
176     shmctl(shared_memory, IPC_RMID, NULL);
177     semctl(sem, BIN, IPC_RMID, 0);
178
179     return 0;
180 }

```

**Результат работы программы:**



```

Producer 1: A
Producer 2: B
Producer 3: C
Consumer 1: A
Consumer 2: B
Consumer 3: C
Producer 1: D
Producer 2: E
Producer 3: F
Consumer 1: D
Consumer 2: E
Consumer 3: F
Producer 1: G
Producer 2: H
Producer 3: I
Consumer 1: G
Consumer 2: H
Consumer 3: I
Producer 1: J
Producer 1: K
Producer 2: L
Producer 2: M
Producer 3: N
Producer 3: O
Consumer 1: J
Consumer 1: K
Consumer 2: L
Consumer 2: M
Consumer 3: N
Consumer 3: O
^Catched

```

**Рисунок 1** – Результат работы программы

## 2 Задание

Написать программу, реализующую задачу «Читатели – писатели» по монитору Хоара с четырьмя функциями: Начать\_чтение, Закончить\_чтение, Начать\_запись, Закончить\_запись. В программе всеми процессами разделяется одно единственное значение в разделяемой памяти. Писатели ее только инкрементируют, читатели могут только читать значение.

Для реализации взаимного исключения используются семафоры

**Программа:**

**Листинг 2** – Задание 2

```

1 #include <stdio.h>
2 #include <stdlib.h>

```

```

3 #include <unistd.h>
4 #include <sys/types.h>
5 #include <sys/ipc.h>
6 #include <sys/sem.h>
7 #include <sys/shm.h>
8 #include <sys/wait.h>
9 #include <sys/stat.h>
10 #include <time.h>
11 #define ACTIVE_WRITER 0
12 #define WAITING_WRITER 1
13 #define ACTIVE_READER 2
14 #define WAITING_READER 3
15
16 int* shared_buffer = NULL;
17
18 struct sembuf startread[5] =
19 {
20     {ACTIVE_WRITER, 0, 0},
21     {WAITING_WRITER, 0, 0},
22     {WAITING_READER, 1, 0},
23     {ACTIVE_READER, 1, 0},
24     {WAITING_READER, -1, 0}
25 };
26 struct sembuf stopread[1] =
27 {
28     {ACTIVE_READER, -1, 0}
29 };
30
31 struct sembuf startwrite[5] =
32 {
33     {ACTIVE_READER, 0, 0},
34     {ACTIVE_WRITER, 0, 0},
35     {WAITING_WRITER, 1, 0},
36     {ACTIVE_WRITER, 1, 0},
37     {WAITING_WRITER, -1, 0}

```

```

38 };
39 struct sembuf stopwrite[1] =
40 {
41     {ACTIVE_WRITER, -1, 0}
42 };
43
44 int perms = S_IRWXU | S_IRWXG | S_IRWXO;
45
46 int readers = 5;
47 int writers = 3;
48
49 int get_sem()
50 {
51     int sem = semget(IPC_PRIVATE, 5, IPC_CREAT | perms);
52     if (sem == -1)
53     {
54         perror("sem\n");
55         exit(1);
56     }
57     int s1 = semctl(sem, ACTIVE_WRITER, SETVAL, 0);
58     int s2 = semctl(sem, ACTIVE_READER, SETVAL, 0);
59     int s3 = semctl(sem, WAITING_WRITER, SETVAL, 0);
60     int s4 = semctl(sem, WAITING_READER, SETVAL, 0);
61
62     return sem;
63 }
64
65 int get_shared_memory()
66 {
67     int shared_memory = shmget(IPC_PRIVATE, sizeof(int),
68                                IPC_CREAT | perms);
69     if (shared_memory == -1)
70     {
71         perror("shmget\n");
72         exit(1);

```

```

72     }
73
74     shared_buffer = (int*)shmat(shared_memory, 0, 0);
75     if (shared_buffer == (int*)-1)
76     {
77         perror("shmat\n");
78         exit(1);
79     }
80
81     *shared_buffer = 0;
82
83     return shared_memory;
84 }
85
86 void start_read(int sem)
87 {
88     if (semop(sem, startread, 5) == -1)
89     {
90         perror("semop\n");
91         exit(1);
92     }
93 }
94
95 void stop_read(int sem)
96 {
97     if (semop(sem, stopread, 1) == -1)
98     {
99         perror("semop\n");
100        exit(1);
101    }
102 }
103
104 void start_write(int sem)
105 {
106     if (semop(sem, startwrite, 5) == -1)

```

```

107     {
108         perror("semop\n");
109         exit(1);
110     }
111 }
112
113 void stop_write(int sem)
114 {
115     if (semop(sem, stopwrite, 1) == -1)
116     {
117         perror("semop\n");
118         exit(1);
119     }
120 }
121
122 void writer(int sem, int id)
123 {
124     while (1)
125     {
126         start_write(sem);
127
128         *shared_buffer += 1;
129         printf("Writer %d: %d\n", id, *shared_buffer);
130
131         stop_write(sem);
132
133         sleep(rand() % 5);
134     }
135 }
136
137 void reader(int sem, int id)
138 {
139     while (1)
140     {
141         start_read(sem);

```

```

142
143     printf("Reader %d: %d\n", id, *shared_buffer);
144
145     stop_read(sem);
146
147     sleep(rand() % 5);
148 }
149 }
150
151 void fork_proc(void (*func)(int sem, int id), int sem, int count)
152 {
153     for (int i = 0; i < count; i++)
154     {
155         pid_t pid = fork();
156
157         if (pid == -1)
158         {
159             perror("fork");
160             exit(1);
161         }
162
163         if (pid == 0)
164         {
165             func(sem, i + 1);
166             exit(0);
167         }
168     }
169 }
170
171 void catch_signal(int signalNum)
172 {
173     printf("Caught\n");
174 }
175
176 int main()

```

```
177 {
178     srand (time (NULL));
179     int shared_memory = get_shared_memory ();
180     int sem = get_sem ();
181
182     fork_proc (writer , sem, writers);
183     fork_proc (reader , sem, readers);
184
185     signal (SIGINT, catch_signal);
186
187     for (int i = 0; i < writers + readers; i++)
188     {
189         int status;
190         wait(&status);
191     }
192
193     shmctl (shared_memory , IPC_RMID , NULL);
194
195     return 0;
196 }
```

**Результат работы программы:**

```
Writer 1: 1
Writer 2: 2
Writer 3: 3
Writer 1: 4
Reader 1: 4
Reader 3: 4
Reader 4: 4
Reader 2: 4
Reader 5: 4
Writer 2: 5
Writer 3: 6
Reader 5: 6
Reader 1: 6
Reader 4: 6
Reader 2: 6
Reader 3: 6
Writer 1: 7
Writer 2: 8
Writer 3: 9
Reader 4: 9
Reader 2: 9
Reader 3: 9
Reader 1: 9
Reader 5: 9
^Catched
```

**Рисунок 2** – Результат работы программы