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		Д.О. Склифасовский
	(Группа)	(Подпись, дата)	(И.О. Фамилия)
Преподаватель			Н.Ю. Рязанова
		(Подпись, дата)	(И.О. Фамилия)

Москва, 2020

Содержание

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 их 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

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

#define SIZE 24

#define FULL 0
#define EMPTY 1
#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("Catched\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
^CCatched
```

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 взаимоисключения используются семафоры

Листинг 2 – Задание 2

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
```

```c
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/shm.h>
#include <sys/wait.h>
#include <sys/stat.h>

#define ACTIVE_WRITER 0
#define WAITING_WRITER 1
#define ACTIVE_READER 2
#define WAITING_READER 3

int* shared_buffer = NULL;

struct sembuf canread[3] =
{
    {ACTIVE_WRITER, 0, 0},
    {WAITING_WRITER, 0, 0},
    {WAITING_READER, 1, 0}
};
struct sembuf startread[2] =
{
    {ACTIVE_READER, 1, 0},
    {WAITING_READER, -1, 0}
};
struct sembuf stopread[1] =
{
    {ACTIVE_READER, -1, 0}
};

struct sembuf canwrite[3] =
{
    {ACTIVE_READER, 0, 0},
    {ACTIVE_WRITER, 0, 0},
    {WAITING_WRITER, 1, 0}
```

```
39 };
40 struct sembuf startwrite[2] =
41 {
42     {ACTIVE_WRITER, 1, 0},
43     {WAITING_WRITER, -1, 0}
44 };
45 struct sembuf stopwrite[1] =
46 {
47     {ACTIVE_WRITER, -1, 0}
48 };
49
50 int perms = S_IRWXU | S_IRWXG | S_IRWXO;
51
52 int readers = 5;
53 int writers = 3;
54
55 int get_sem()
56 {
57     int sem = semget(IPC_PRIVATE, 5, IPC_CREAT | perms);
58     if (sem == -1)
59     {
60         perror("sem\n");
61         exit(1);
62     }
63     int s1 = semctl(sem, ACTIVE_WRITER, SETVAL, 0);
64     int s2 = semctl(sem, ACTIVE_READER, SETVAL, 0);
65     int s3 = semctl(sem, WAITING_WRITER, SETVAL, 0);
66     int s4 = semctl(sem, WAITING_READER, SETVAL, 0);
67
68     return sem;
69 }
70
71 int get_shared_memory()
72 {
73     int shared_memory = shmget(IPC_PRIVATE, sizeof(int),
```

```
        IPC_CREAT | perms);
74      if (shared_memory == -1)
75      {
76          perror("shmget\n");
77          exit(1);
78      }
79
80      shared_buffer = (int*)shmat(shared_memory, 0, 0);
81      if (shared_buffer == (int*)-1)
82      {
83          perror("shmat\n");
84          exit(1);
85      }
86
87      *shared_buffer = 0;
88
89      return shared_memory;
90  }
91
92  void start_read(int sem)
93  {
94      if (semop(sem, canread, 3) == -1)
95      {
96          perror("semop\n");
97          exit(1);
98      }
99      if (semop(sem, startread, 2) == -1)
100     {
101         perror("semop\n");
102         exit(1);
103     }
104 }
105
106 void stop_read(int sem)
107 {
```

```
108     if (semop(sem, stopread, 1) == -1)
109     {
110         perror("semop\n");
111         exit(1);
112     }
113 }
114
115 void start_write(int sem)
116 {
117     if (semop(sem, canwrite, 3) == -1)
118     {
119         perror("semop\n");
120         exit(1);
121     }
122     if (semop(sem, startwrite, 2) == -1)
123     {
124         perror("semop\n");
125         exit(1);
126     }
127 }
128
129 void stop_write(int sem)
130 {
131     if (semop(sem, stopwrite, 1) == -1)
132     {
133         perror("semop\n");
134         exit(1);
135     }
136 }
137
138 void writer(int sem, int id)
139 {
140     while (1)
141     {
142         start_write(sem);
```

```
143
144             *shared_buffer += 1;
145             printf("Writer %d: %d\n", id, *shared_buffer);
146
147             stop_write(sem);
148
149             sleep(rand() % 5);
150         }
151 }
152
153 void reader(int sem, int id)
154 {
155         while (1)
156         {
157             start_read(sem);
158
159             printf("Reader %d: %d\n", id, *shared_buffer);
160
161             stop_read(sem);
162
163             sleep(rand() % 5);
164         }
165 }
166
167 void fork_proc(void (*func)(int sem, int id), int sem, int count)
168 {
169         for (int i = 0; i < count; i++)
170         {
171             pid_t pid = fork();
172
173             if (pid == -1)
174             {
175                 perror("fork");
176                 exit(1);
177             }
```

```
178
179             if (pid == 0)
180             {
181                 func(sem, i + 1);
182                 exit(0);
183             }
184         }
185 }
186
187 void catch_signal(int signalNum)
188 {
189     printf("Catched\n");
190 }
191
192 int main()
193 {
194     int shared_memory = get_shared_memory();
195     int sem = get_sem();
196
197     fork_proc(writer, sem, writers);
198     fork_proc(reader, sem, readers);
199
200     signal(SIGINT, catch_signal);
201
202     for (int i = 0; i < writers + readers; i++)
203     {
204         int status;
205         wait(&status);
206     }
207
208     shmctl(shared_memory, IPC_RMID, NULL);
209
210     return 0;
211 }
```

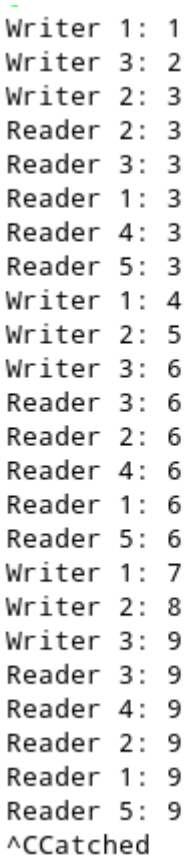

Рисунок 2 – Результат работы программы