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.В. Сусликов

(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель _____ Н.Ю. Рязанова

(Подпись, дата) (И.О. Фамилия)

Москва, 2020

Содержание

Задание 1

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 1

В Листинге 1 описан код программы.

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  #define LEN 26
22
23  char* shared_buffer = NULL;
24  char* consumer_index = 0;
25  char* producer_index = 0;
26
27  char alph[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
28
29  struct sembuf start_producer[2] = {
30    {EMPTY, -1, 0},
31    {BIN, -1, 0}
32  };
33  struct sembuf stop_producer[2] = {
34    {BIN, 1, 0},
35    {FULL, 1, 0}
36  };
37
38  struct sembuf start_consumer[2] = {
39    {FULL, -1, 0},
40    {BIN, -1, 0}
41  };
42  struct sembuf stop_consumer[2] = {
43    {BIN, 1, 0},
44    {EMPTY, 1, 0}
45  };
46
47  int perms = S_IRWXU | S_IRWXG | S_IRWXO;
48
```

```
49  void error_check(int fd, char* msg)
50  {
51    if (fd == -1)
52    {
53      perror(msg);
54      exit(1);
55    }
56  }
57
58  int get_sem()
59  {
60    int sem = semget(IPC_PRIVATE, 3, IPC_CREAT | perms);
61    error_check(sem, "sem");
62
63    int s1 = semctl(sem, FULL, SETVAL, 0);
64    error_check(s1, "semctl");
65
66    int s2 = semctl(sem, EMPTY, SETVAL, SIZE);
67    error_check(s2, "semctl");
68
69    int s3 = semctl(sem, BIN, SETVAL, 1);
70    error_check(s3, "semctl");
71
72    return sem;
73  }
74
75  int get_shared_memory()
76  {
77    int shared_memory = shmget(IPC_PRIVATE, (SIZE + 2) * sizeof(char
        ), IPC_CREAT | perms);
78    error_check(shared_memory, "shmget");
79
80    producer_index = (char*)shmat(shared_memory, 0, 0);
81    if (producer_index == (char*)-1)
82    {
```

```
83        perror("shmat\n");
84        exit(1);
85      }
86      consumer_index = producer_index + sizeof(char);
87      shared_buffer = consumer_index + sizeof(char);
88
89      return shared_memory;
90    }
91
92    void producer(int sem, int id)
93    {
94      while(1)
95      {
96        error_check(semop(sem, start_producer, 2), "semop");
97
98        shared_buffer[*producer_index] = alph[*producer_index];
99        printf("Producer  %d: %c\n", id, alph[*producer_index]);
100
101       (*producer_index)++;
102       if (*producer_index == LEN)
103       *producer_index = 0;
104
105       error_check(semop(sem, stop_producer, 2), "semop");
106
107       sleep(rand() % 5);
108     }
109   }
110
111   void consumer(int sem, int id)
112   {
113     while(1)
114     {
115       error_check(semop(sem, start_consumer, 2), "semop");
116
117       printf("Consumer  %d: %c\n", id, shared_buffer[*consumer_index
```

```
        ]);
118
119     (*consumer_index)++;
120     if (*consumer_index == LEN)
121     *consumer_index = 0;
122
123     error_check(semop(sem, stop_consumer, 2), "semop");
124
125     sleep(rand() % 5 + 1);
126   }
127 }
128
129 void catch_signal(int signalNum)
130 {
131   printf("Signal catched\n");
132 }
133
134 int main()
135 {
136   srand(time(NULL));
137   int shared_memory = get_shared_memory();
138   int sem = get_sem();
139
140   for (int i = 0; i < PROD; i++)
141   {
142     pid_t pid = fork();
143     error_check(pid, "fork");
144
145     if (pid == 0)
146     {
147       producer(sem, i + 1);
148       exit(0);
149     }
150   }
151
```

```
152     for (int i = 0; i < CONS; i++)
153     {
154       pid_t pid = fork();
155       error_check(pid, "fork");
156 
157       if (pid == 0)
158       {
159         consumer(sem, i + 1);
160         exit(0);
161       }
162     }
163 
164     signal(SIGINT, catch_signal);
165 
166     for (int i = 0; i < (PROD + CONS); i++)
167     {
168       int status;
169       wait(&status);
170     }
171 
172     shmctl(shared_memory, IPC_RMID, NULL);
173     semctl(sem, BIN, IPC_RMID, 0);
174 
175     return 0;
176   }
```

Ниже на Рисунке 1 показан пример работы данной программы.

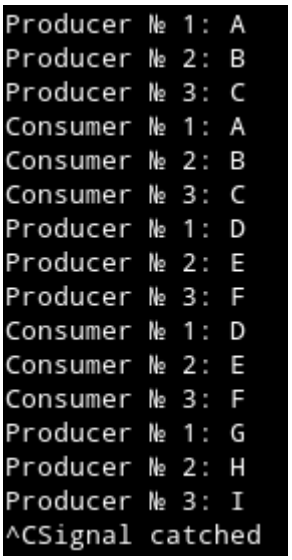

Рисунок 1 – Пример работы Программы 1

Задание 2

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

Программа 2

В Листинге 2 показан код программы.

```
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 ACTIVE_WRITER 0
13 #define WAITING_WRITER 1
14 #define ACTIVE_READER 2
15 #define WAITING_READER 3
16
17 #define WRITERS 3
18 #define READERS 5
19
20 int* shared_buffer = NULL;
```

```
21
22  struct sembuf start_reading[5] =
23  {
24    {WAITING_READER, 1, 0},
25    {ACTIVE_WRITER, 0, 0},
26    {WAITING_WRITER, 0, 0},
27    {ACTIVE_READER, 1, 0},
28    {WAITING_READER, -1, 0}
29  };
30  struct sembuf stop_reading[1] =
31  {
32    {ACTIVE_READER, -1, 0}
33  };
34
35  struct sembuf start_writing[5] =
36  {
37    {WAITING_WRITER, 1, 0},
38    {ACTIVE_READER, 0, 0},
39    {ACTIVE_WRITER, 0, 0},
40    {ACTIVE_WRITER, 1, 0},
41    {WAITING_WRITER, -1, 0}
42  };
43  struct sembuf stop_writing[1] =
44  {
45    {ACTIVE_WRITER, -1, 0}
46  };
47
48  int perms = S_IRWXU | S_IRWXG | S_IRWXO;
49
50  void error_check(int fd, char* msg)
51  {
52    if (fd == -1)
53    {
54      perror(msg);
55      exit(1);
```

```
56      }
57    }
58
59    int get_sem()
60    {
61      int sem = semget(IPC_PRIVATE, 5, IPC_CREAT | perms);
62      error_check(sem, "sem");
63
64      int s1 = semctl(sem, ACTIVE_WRITER, SETVAL, 0);
65      error_check(s1, "semctl");
66
67      int s2 = semctl(sem, ACTIVE_READER, SETVAL, 0);
68      error_check(s2, "semctl");
69
70      int s3 = semctl(sem, WAITING_WRITER, SETVAL, 0);
71      error_check(s3, "semctl");
72
73      int s4 = semctl(sem, WAITING_READER, SETVAL, 0);
74      error_check(s4, "semctl");
75
76      return sem;
77    }
78
79    int get_shared_memory()
80    {
81      int shared_memory = shmget(IPC_PRIVATE, sizeof(int), IPC_CREAT |
          perms);
82      error_check(shared_memory, "shmget");
83
84      shared_buffer = (int*)shmat(shared_memory, 0, 0);
85      if (shared_buffer == (int*)-1)
86      {
87        perror("shmat\n");
88        exit(1);
89      }
```

```
90
91      *shared_buffer = 0;
92
93      return shared_memory;
94    }
95
96    void writer(int sem, int id)
97    {
98      while (1)
99      {
100       error_check(semop(sem, start_writing, 5), "semop");
101
102       *shared_buffer += 1;
103       printf("Writer  %d: %d\n", id, *shared_buffer);
104
105       error_check(semop(sem, stop_writing, 1), "semop");
106
107       sleep(rand() % 5);
108     }
109   }
110
111   void reader(int sem, int id)
112   {
113     while (1)
114     {
115       error_check(semop(sem, start_reading, 5), "semop");
116
117       printf("Reader  %d: %d\n", id, *shared_buffer);
118
119       error_check(semop(sem, stop_reading, 1), "semop");
120
121       sleep(rand() % 5);
122     }
123   }
124
```

```
125  void catch_signal(int signalNum)
126  {
127    printf("Signal catched\n");
128  }
129
130  int main()
131  {
132    srand(time(NULL));
133    int shared_memory = get_shared_memory();
134    int sem = get_sem();
135
136    for (int i = 0; i < WRITERS; i++)
137    {
138      pid_t pid = fork();
139      error_check(pid, "fork");
140
141      if (pid == 0)
142      {
143        writer(sem, i + 1);
144        exit(0);
145      }
146    }
147
148    for (int i = 0; i < READERS; i++)
149    {
150      pid_t pid = fork();
151      error_check(pid, "fork");
152
153      if (pid == 0)
154      {
155        reader(sem, i + 1);
156        exit(0);
157      }
158    }
159
```

```
160     signal(SIGINT, catch_signal);
161
162     for (int i = 0; i < (WRITERS + READERS); i++)
163     {
164       int status;
165       wait(&status);
166     }
167
168     shmctl(shared_memory, IPC_RMID, NULL);
169
170     return 0;
171   }
```

Ниже на Рисунке 2 показан пример работы данной программы.

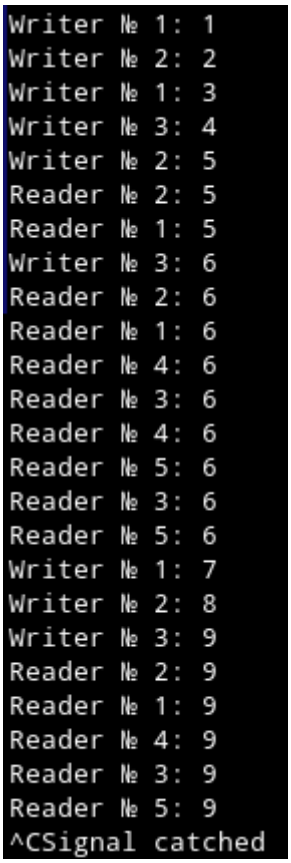

Рисунок 2 – Пример работы Программы 2