

МГТУ им. БАУМАНА

ЛАБОРАТОРНАЯ РАБОТА №5

По курсу: "ОПЕРАЦИОННЫЕ СИСТЕМЫ"

Взаимодействие параллельных процессов.

Работу выполнил: Мокеев Даниил, ИУ7-56

Преподаватель: Рязанова Н.Ю.

Москва, 2020

0.1 Листинг кода алгоритмов

В данном разделе будут приведены листинги кода реализованных программ.

Листинг 1: Задача производства-потребления

```
1 #include <signal.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <sys/stat.h>
5 #include <sys/sem.h>
6 #include <sys/shm.h>
7 #include <time.h>
8 #include <unistd.h>
9 #include <sys/wait.h>
10
11 #define SEM_BIN    0
12 #define SEM_EMPTY  1
13 #define SEM_FULL   2
14
15 #define P  -1
16 #define V  1
17
18 #define PRODUCERS_COUNT 3
19 #define CONSUMERS_COUNT 3
20
21 #define BUFFER_SIZE 3
22
23 #define PERMS S_IRWXU | S_IRWXG | S_IRWXO //permission to read, write & ↵
24     execute by user, group & others
25 #define CONSUMER_BORDER "\t\t\t\t\t"
26
27 int sem_id = -1;
28 int shm_id = -1;
29
30 int *shm = NULL;
31 int *shm_pos = NULL;
32
33 struct sembuf producer_start[2] = {
34     {SEM_EMPTY, P, SEM_UNDO},
35     {SEM_BIN,   P, SEM_UNDO}
36 };
37 struct sembuf producer_stop[2] = {
38     {SEM_BIN,   V, SEM_UNDO},
39     {SEM_FULL,  V, SEM_UNDO}
40 };
41
42 struct sembuf consumer_start[2] = {
43     {SEM_FULL,  P, SEM_UNDO},
```

```

44  {SEM_BIN,    P, SEM_UNDO}
45  };
46  struct sembuf consumer_stop[2] = {
47      {SEM_BIN,    V, SEM_UNDO},
48      {SEM_EMPTY, V, SEM_UNDO}
49  };
50
51  void fork_children(const int n, void (*func)(const int)) {
52      for (int i = 0; i < n; ++i) {
53          const pid_t pid = fork();
54
55          if (pid == -1) {
56              perror("fork");
57              exit(1);
58          } else if (pid == 0) {
59              if (func) {
60                  func(i);
61              }
62              exit(1);
63          }
64      }
65  }
66
67  void wait_children(const int n) {
68      for (int i = 0; i < n; ++i) {
69          int status;
70          const pid_t child_pid = wait(&status);
71          if (child_pid == -1) {
72              perror("wait error");
73              exit(1);
74          }
75
76          if (WIFEXITED(status)) {
77              printf("Process %d returns %d\n", child_pid, WEXITSTATUS(status));
78          } else if (WIFSIGNALED(status)) {
79              printf("Process %d terminated with signal %d\n", child_pid, ↵
                  WTERMSIG(status));
80          } else if (WIFSTOPPED(status)) {
81              printf("Process %d stopped due signal %d\n", child_pid, WSTOPSIG(↵
                  status));
82          }
83      }
84  }
85
86  void producer(const int id) {
87      for (char i = 97; i <= 122; i++) {
88          sleep(rand() % 3);
89          if (semop(sem_id, producer_start, 2) == -1) {
90              perror("semop");

```

```

91     exit(1);
92 }
93 // write next value in shared memory
94 *(shm + *shm_pos) = i;
95 printf("Producer %d (pid %d) produces %c\n", id, getpid(), i);
96 (*shm_pos)++;
97
98 if (semop(sem_id, producer_stop, 2) == -1) {
99     perror("semop");
100    exit(1);
101 }
102
103 if (i == 122)
104    exit(0);
105 }
106 }
107
108 void consumer(const int id) {
109     while(1) {
110         sleep(rand() % 2);
111         if (semop(sem_id, consumer_start, 2) == -1) {
112             perror("semop");
113             exit(1);
114         }
115
116         printf(CONSUMER_BORDER"Consumer %d (pid %d) consumes %c\n", id, ↵
117             getpid(), *(shm + (*shm_pos)-1));
118         (*shm_pos)--;
119
120         if (semop(sem_id, consumer_stop, 2) == -1) {
121             perror("semop");
122             exit(1);
123         }
124
125         if (*(shm + (*shm_pos)) == 122)
126             exit(0);
127     }
128 }
129 }
130
131
132 void init_semaphores() {
133     sem_id = semget(IPC_PRIVATE, 3, IPC_CREAT | PERMS);
134     if (sem_id == -1) {
135         perror("semget");
136         exit(1);
137     }
138     if (semctl(sem_id, SEM_BIN, SETVAL, 1) == -1 ||

```

```

139     semctl(sem_id, SEM_EMPTY, SETVAL, BUFFER_SIZE) == -1 ||
140     semctl(sem_id, SEM_FULL, SETVAL, 0) == -1) {
141         perror("semctl");
142         exit(1);
143     }
144 }
145
146 void init_shared_memory() {
147     shm_id = shmget(IPC_PRIVATE, (BUFFER_SIZE+2) * sizeof(int), IPC_CREAT ↵
        | PERMS);
148
149     if (shm_id == -1) {
150         perror("shmget");
151         exit(1);
152     }
153     shm = shmat(shm_id, 0, 0);
154     if (shm == (void *) -1) {
155         perror("shmat");
156         exit(1);
157     }
158
159     shm_pos = shm;
160     *shm_pos = 0;
161     shm = shm + 2;
162 }
163
164 int main() {
165     int children = 0;
166     srand((unsigned int) time(NULL));
167
168     init_semaphores();
169     init_shared_memory();
170
171     fork_children(PRODUCERS_COUNT, producer);
172     fork_children(CONSUMERS_COUNT, consumer);
173
174     wait_children(PRODUCERS_COUNT + CONSUMERS_COUNT);
175
176     shmctl(shm_id, IPC_RMID, NULL);
177     semctl(sem_id, SEM_BIN, IPC_RMID, 0);
178 }

```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Producer 0 (pid 243081) produces a
Producer 1 (pid 243082) produces a
Producer 2 (pid 243083) produces a
Consumer 0 (pid 243084) consumes a
Producer 0 (pid 243081) produces b
Consumer 1 (pid 243085) consumes b
Consumer 2 (pid 243086) consumes a
Producer 1 (pid 243082) produces b
Producer 2 (pid 243083) produces b
Consumer 0 (pid 243084) consumes b
Producer 0 (pid 243081) produces c
Consumer 0 (pid 243084) consumes c
Producer 1 (pid 243082) produces c
Consumer 0 (pid 243084) consumes c
Producer 2 (pid 243083) produces c
Consumer 1 (pid 243085) consumes c
Producer 0 (pid 243081) produces d
Consumer 0 (pid 243084) consumes d
Producer 1 (pid 243082) produces d
Consumer 1 (pid 243085) consumes d
Producer 2 (pid 243083) produces d
Consumer 1 (pid 243085) consumes d
Consumer 2 (pid 243086) consumes b
Consumer 1 (pid 243085) consumes a
Producer 0 (pid 243081) produces e
Consumer 2 (pid 243086) consumes e
Producer 0 (pid 243081) produces f
Consumer 0 (pid 243084) consumes f
Producer 1 (pid 243082) produces e
Consumer 1 (pid 243085) consumes e
Producer 2 (pid 243083) produces e
Consumer 2 (pid 243086) consumes e
Producer 1 (pid 243082) produces f
Consumer 0 (pid 243084) consumes f
Producer 2 (pid 243083) produces f
Consumer 1 (pid 243085) consumes f
Producer 0 (pid 243081) produces g
Consumer 2 (pid 243086) consumes g
Producer 1 (pid 243082) produces g
Consumer 0 (pid 243084) consumes g
Producer 2 (pid 243083) produces g
Consumer 1 (pid 243085) consumes g
Producer 0 (pid 243081) produces h
Producer 1 (pid 243082) produces h
Consumer 0 (pid 243084) consumes h
Producer 2 (pid 243083) produces h
```

Рис. 1: Пример работы программы. BUFFER_SIZE = 3 (начало вывода)

```

Producer 1 (pid 243082) produces v
Producer 0 (pid 243081) produces w
Producer 2 (pid 243083) produces w
Producer 1 (pid 243082) produces w
Producer 0 (pid 243081) produces x
Producer 2 (pid 243083) produces x
Producer 1 (pid 243082) produces x
Producer 2 (pid 243083) produces y
Producer 0 (pid 243081) produces y
Producer 1 (pid 243082) produces y
Producer 2 (pid 243083) produces z
Producer 0 (pid 243081) produces z
Producer 1 (pid 243082) produces z
Process 243083 returns 0
Process 243081 returns 0
Process 243082 returns 0
Process 243084 returns 0
Process 243085 returns 0
Process 243086 returns 0
[1] + Done
Consumer 0 (pid 243084) consumes v
Consumer 1 (pid 243085) consumes v
Consumer 0 (pid 243084) consumes w
Consumer 1 (pid 243085) consumes w
Consumer 2 (pid 243086) consumes w
Consumer 2 (pid 243086) consumes x
Consumer 0 (pid 243084) consumes x
Consumer 1 (pid 243085) consumes x
Consumer 0 (pid 243084) consumes y
Consumer 1 (pid 243085) consumes y
Consumer 2 (pid 243086) consumes y
Consumer 0 (pid 243084) consumes z
Consumer 1 (pid 243085) consumes z
Consumer 2 (pid 243086) consumes z
"/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm} 0<"/tmp/Microsoft-M

```

Рис. 2: Пример работы программы. BUFFER_SIZE = 3 (конец вывода)

```
1 #include <sys/shm.h>
2 #include <sys/sem.h>
3 #include <fcntl.h>
4 #include <unistd.h>
5 #include <sys/wait.h>
6 #include <stdio.h>
7 #include <stdlib.h>
8 #include <signal.h>
9 #include <time.h>
10
11 #define READER_BORDER "\t\t\t\t\t"
12
13 #define PERMS S_IRWXU | S_IRWXG | S_IRWXO
14
15 #define MAX_VALUE 10
16 #define WRITERS 3
17 #define READERS 5
18
19 #define ACTIVE_WR 0
20 #define ACTIVE_RR 1
21 #define WAITING_W 2
22 #define WAITING_R 3
23
24 #define V 1
25 #define P -1
26 #define Z 0
27
28 #define SEM_ACTIVE_WR 0
29 #define SEM_ACTIVE_RR 1
30 #define SEM_WAITING_W 2
31 #define SEM_WAITING_R 3
32
33 struct sembuf start_reading[] = {
34     {SEM_WAITING_R, V, SEM_UNDO},
35     {SEM_ACTIVE_WR, Z, SEM_UNDO},
36     {SEM_WAITING_W, Z, SEM_UNDO},
37     {SEM_ACTIVE_RR, V, SEM_UNDO},
38     {SEM_WAITING_R, P, SEM_UNDO}
39 };
40 struct sembuf stop_read[] = {
41     {SEM_ACTIVE_RR, P, SEM_UNDO}
42 };
43
44 struct sembuf start_writing[] = {
45     {SEM_WAITING_W, V, SEM_UNDO},
46     {SEM_ACTIVE_RR, Z, SEM_UNDO},
47     {SEM_ACTIVE_WR, Z, SEM_UNDO},
48     {SEM_ACTIVE_WR, V, SEM_UNDO},
```



```

49     {SEM_WAITING_W, P, SEM_UNDO}
50 };
51 struct sembuf stop_write[] = {
52     {SEM_ACTIVE_WR, P, SEM_UNDO}
53 };
54
55 int *shared_value;
56
57 pid_t *child_pids;
58
59
60 void kill_writers() {
61     for (int i = 0; i < WRITERS; i++) {
62         if (child_pids[i] == getpid()) {
63             continue;
64         }
65         kill(child_pids[i], SIGTERM);
66     }
67 }
68
69 int init_sh_mem() {
70     int fd = shmget(IPC_PRIVATE, sizeof(int), IPC_CREAT | PERMS);
71     if (fd == -1) {
72         perror("shmget");
73         exit(1);
74     }
75     return fd;
76 }
77
78 int *get_sh_mem_addr(const int fd) {
79     int *addr = (int *) shmat(fd, NULL, 0);
80     if (addr == (int *) -1) {
81         perror("shmat error");
82         exit(1);
83     }
84     return addr;
85 }
86
87 int init_semaphores() {
88     int semid = semget(IPC_PRIVATE, 4, IPC_CREAT | PERMS);
89     if (semid == -1) {
90         perror("semget error");
91         exit(1);
92     }
93
94     int mem_ctrl = semctl(semid, WAITING_R, SETVAL, 0);
95     int writer_ctrl = semctl(semid, ACTIVE_WR, SETVAL, 0);
96     int reader_ctrl = semctl(semid, ACTIVE_RR, SETVAL, 0);
97     int wait_ctrl = semctl(semid, WAITING_W, SETVAL, 0);

```

```

98
99  if (mem_ctrl == -1 || writer_ctrl == -1 || reader_ctrl == -1 || ↵
    wait_ctrl == -1) {
100      perror("semctl");
101      exit(1);
102  }
103
104  return semid;
105 }
106
107 void writer(int semid, int number) {
108     int can = semop(semid, start_writing, 5);
109     if (can == -1) {
110         perror("semop start writing error");
111         exit(1);
112     }
113     // Z condition for stopping all writers
114     if (*shared_value >= MAX_VALUE) {
115         kill_writers();
116         int sem_op_stop = semop(semid, stop_write, 1);
117         if (sem_op_stop == -1) {
118             perror("semop stop write error");
119             exit(1);
120         }
121         exit(0);
122     }
123     // write
124     (*shared_value)++;
125     printf("Writer #%d, pid=%d wrote value %d\n", number, getpid(), *↵
        shared_value);
126
127     int sem_op_stop = semop(semid, stop_write, 1);
128     if (sem_op_stop == -1) {
129         perror("semop stop_write");
130         exit(1);
131     }
132
133     sleep(rand() % 10);
134 }
135
136 void reader(int semid, int number) {
137     int can = semop(semid, start_reading, 5);
138     if (can == -1) {
139         perror("semop start_reading");
140         exit(1);
141     }
142     // read
143     int val = *shared_value;
144     printf(READER_BORDER"Reader #%d, pid=%d read value: %d\n", number, ↵

```

```

        getpid(), val);
145
146 int sem_op_stop = semop(semid, stop_read, 1);
147 if (sem_op_stop == -1) {
148     perror("semop stop_read");
149     exit(1);
150 }
151 // Z condition for stopping all readers
152 if (val >= MAX_VALUE) {
153     exit(0);
154 }
155
156 sleep(rand() % 10);
157 }
158
159 void init_writer(int number, int sem_id) {
160     pid_t pid;
161
162     if ((pid = fork()) == -1) {
163         printf("Can't fork");
164         exit(1);
165     }
166
167     if (pid == 0) {
168         printf("Writer #%d created, pid: %d\n", number, getpid());
169         while (1) {
170             writer(sem_id, number);
171         }
172     }
173     else {
174         child_pids[number] = pid;
175     }
176 }
177
178 void init_reader(int number, const int sem_id) {
179     pid_t pid;
180
181     if ((pid = fork()) == -1) {
182         printf("Can't fork");
183         exit(1);
184     }
185
186     if (pid == 0) {
187         printf(READER_BORDER"Reader #%d created, pid: %d\n", number, getpid()↵
            ());
188         while (1) {
189             reader(sem_id, number);
190         }
191     }

```

```

192     else {
193         child_pids[WRITERS + number] = pid;
194     }
195 }
196
197 int main() {
198     int sh_mem_fd = init_sh_mem();
199     int *sh_mem = get_sh_mem_addr(sh_mem_fd);
200
201     shared_value = sh_mem;
202     *shared_value = 0;
203     child_pids = shared_value + 1;
204
205     int semid = init_semaphores();
206
207     for (int i = 0; i < WRITERS; i++) {
208         init_writer(i, semid);
209     }
210
211     for (int i = 0; i < READERS; i++) {
212         init_reader(i, semid);
213     }
214
215     for (int i = 0; i < WRITERS + READERS; i++) {
216         int *status;
217         wait(status);
218     }
219 }

```

```

Writer #0 created, pid: 225723
Writer #0, pid=225723 wrote value 1
Writer #1 created, pid: 225724
Writer #1, pid=225724 wrote value 2
Writer #2 created, pid: 225725
Writer #2, pid=225725 wrote value 3

Reader #0 created, pid: 225726
Reader #0, pid=225726 read value: 3
Reader #1 created, pid: 225727
Reader #1, pid=225727 read value: 3
Reader #2 created, pid: 225728
Reader #2, pid=225728 read value: 3
Reader #3 created, pid: 225729
Reader #3, pid=225729 read value: 3
Reader #4 created, pid: 225730
Reader #4, pid=225730 read value: 3

Writer #0, pid=225723 wrote value 4
Writer #1, pid=225724 wrote value 5
Writer #2, pid=225725 wrote value 6

Reader #0, pid=225726 read value: 6
Reader #1, pid=225727 read value: 6
Reader #2, pid=225728 read value: 6
Reader #3, pid=225729 read value: 6
Reader #4, pid=225730 read value: 6

Writer #0, pid=225723 wrote value 7
Writer #1, pid=225724 wrote value 8
Writer #2, pid=225725 wrote value 9

Reader #0, pid=225726 read value: 9
Reader #1, pid=225727 read value: 9
Reader #2, pid=225728 read value: 9
Reader #3, pid=225729 read value: 9
Reader #4, pid=225730 read value: 9

Writer #0, pid=225723 wrote value 10

Reader #0, pid=225726 read value: 10
Reader #1, pid=225727 read value: 10
Reader #2, pid=225728 read value: 10
Reader #3, pid=225729 read value: 10
Reader #4, pid=225730 read value: 10

[1] + Done
danya@danya-K13MM3:~/Рабочий стол/operating-system/sem 5/lab05$

```

Рис. 3: Пример работы программы.