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

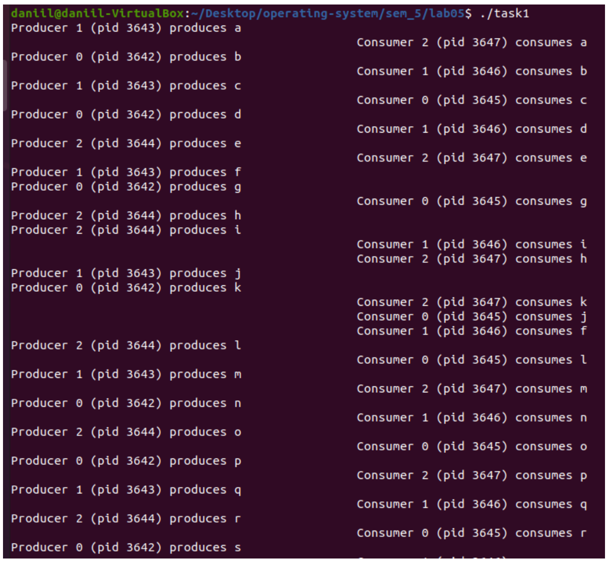
В данном разделе будут приведены листинги кода реализованных программ.

Листинг 1: Задача производства-потребления

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

#define SEM_BIN   0
#define SEM_EMPTY 1
#define SEM_FULL  2

#define P -1
#define V 1

#define PRODUCERS_COUNT 3
#define CONSUMERS_COUNT 3

#define BUFFER_SIZE 3

#define PERMS S_IRWXU | S_IRWXG | S_IRWXO //permition to read, write & execute by user, group & others

#define COMSUMER_BORDER "\t\t\t\t\t\t"

int sem_id = -1;
int shm_id = -1;

int *shm = NULL;
int *shm_pos = NULL;
int *produced_value = NULL;
int *last_consumed = NULL;

struct sembuf producer_start[2] = {
  {SEM_EMPTY, P, SEM_UNDO},
  {SEM_BIN,   P, SEM_UNDO}
};
struct sembuf producer_stop[2] = {
  {SEM_BIN,   V, SEM_UNDO},
  {SEM_FULL,  V, SEM_UNDO}
};

```

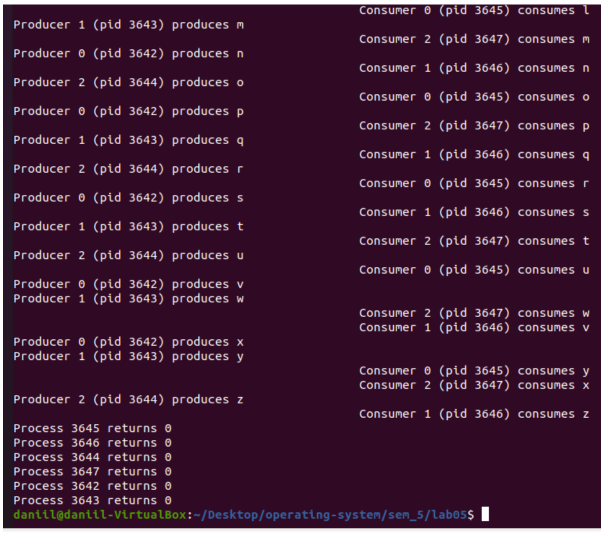
```c
struct sembuf consumer_start[2] = {
  {SEM_FULL,  P, SEM_UNDO},
  {SEM_BIN,   P, SEM_UNDO}
};
struct sembuf consumer_stop[2] = {
  {SEM_BIN,   V, SEM_UNDO},
  {SEM_EMPTY, V, SEM_UNDO}
};

void fork_children(const int n, void (*func)(const int)) {
  for (int i = 0; i < n; ++i) {
    const pid_t pid = fork();

    if (pid == -1) {
      perror("fork");
      exit(1);
    } else if (pid == 0) {
      if (func) {
        func(i);
      }
      exit(1);
    }
  }
}

void wait_children(const int n) {
  for (int i = 0; i < n; ++i) {
    int status;
    const pid_t child_pid = wait(&status);
    if (child_pid == -1) {
      perror("wait error");
      exit(1);
    }

    if (WIFEXITED(status)) {
      printf("Process %d returns %d\n", child_pid, WEXITSTATUS(status));
    } else if (WIFSIGNALED(status)) {
      printf("Process %d terminated with signal %d\n", child_pid, WTERMSIG(status));
    } else if (WIFSTOPPED(status)) {
      printf("Process %d stopped due signal %d\n", child_pid, WSTOPSIG(status));
    }
  }
}

void producer(const int id) {
  while(1) {
    sleep(rand() % 3);

    if(*produced_value > 122)
    exit(0);

    if (semop(sem_id, producer_start, 2) == -1) {
      perror("semop");
      exit(1);
    }

    // write next value in shared memory
    *(shm + *shm_pos) = *produced_value;
    printf("Producer %d (pid %d) produces %c\n", id, getpid(), *↩
       produced_value);
    (*shm_pos)++;
    (*produced_value)++;

    if (semop(sem_id, producer_stop, 2) == -1) {
      perror("semop");
      exit(1);
    }

  }
}

void consumer(const int id) {
  while(1) {
    sleep(rand() % 2);

    if (*last_consumed >= 122)
    exit(0);

    if (semop(sem_id, consumer_start, 2) == -1) {
      perror("semop");
      exit(1);
    }

    printf(COMSUMER_BORDER"Consumer %d (pid %d) consumes %c\n", id, ↩
       getpid(), *(shm +(*shm_pos)-1));
    *(last_consumed) = *(shm +(*shm_pos)-1);
    (*shm_pos)--;

    if (semop(sem_id, consumer_stop, 2) == -1) {
      perror("semop");
      exit(1);
    }
  }
}


```

```c
void init_semaphores() {
  sem_id = semget(IPC_PRIVATE, 3, IPC_CREAT | PERMS);
  if (sem_id == -1) {
    perror("semget");
    exit(1);
  }
  if (semctl(sem_id, SEM_BIN,   SETVAL, 1) == -1 ||
  semctl(sem_id, SEM_EMPTY, SETVAL, BUFFER_SIZE) == -1 ||
  semctl(sem_id, SEM_FULL,  SETVAL, 0) == -1) {
    perror("semctl");
    exit(1);
  }
}

void init_shared_memory() {
  shm_id = shmget(IPC_PRIVATE, (BUFFER_SIZE+3) * sizeof(int), IPC_CREAT | PERMS);

  if (shm_id == -1) {
    perror("shmget");
    exit(1);
  }
  shm = shmat(shm_id, 0, 0);
  if (shm == (void *) -1) {
    perror("shmat");
    exit(1);
  }

  shm_pos = shm;
  *shm_pos = 0;
  produced_value = shm+2;
  last_consumed = shm+1;

  *produced_value = 97;
  *last_consumed = 97;
  shm = shm + 3;

}

int main() {
  int children = 0;
  srand((unsigned int) time(NULL));

  init_semaphores();
  init_shared_memory();

  fork_children(PRODUCERS_COUNT, producer);
  fork_children(CONSUMERS_COUNT, consumer);

  wait_children(PRODUCERS_COUNT + CONSUMERS_COUNT);

  shmctl(shm_id, IPC_RMID, NULL);
  semctl(sem_id, SEM_BIN, IPC_RMID, 0);
}
```



```
daniil@daniil-VirtualBox:~/Desktop/operating-system/sem_5/lab05$ ./task1
Producer 1 (pid 3643) produces a
                                                Consumer 2 (pid 3647) consumes a
Producer 0 (pid 3642) produces b
                                                Consumer 1 (pid 3646) consumes b
Producer 1 (pid 3643) produces c
                                                Consumer 0 (pid 3645) consumes c
Producer 0 (pid 3642) produces d
                                                Consumer 1 (pid 3646) consumes d
Producer 2 (pid 3644) produces e
                                                Consumer 2 (pid 3647) consumes e
Producer 1 (pid 3643) produces f
Producer 0 (pid 3642) produces g
                                                Consumer 0 (pid 3645) consumes g
Producer 2 (pid 3644) produces h
Producer 2 (pid 3644) produces i
                                                Consumer 1 (pid 3646) consumes i
                                                Consumer 2 (pid 3647) consumes h
Producer 1 (pid 3643) produces j
Producer 0 (pid 3642) produces k
                                                Consumer 2 (pid 3647) consumes k
                                                Consumer 0 (pid 3645) consumes j
                                                Consumer 1 (pid 3646) consumes f
Producer 2 (pid 3644) produces l
                                                Consumer 0 (pid 3645) consumes l
Producer 1 (pid 3643) produces m
                                                Consumer 2 (pid 3647) consumes m
Producer 0 (pid 3642) produces n
                                                Consumer 1 (pid 3646) consumes n
Producer 2 (pid 3644) produces o
                                                Consumer 0 (pid 3645) consumes o
Producer 0 (pid 3642) produces p
                                                Consumer 2 (pid 3647) consumes p
Producer 1 (pid 3643) produces q
                                                Consumer 1 (pid 3646) consumes q
Producer 2 (pid 3644) produces r
                                                Consumer 0 (pid 3645) consumes r
Producer 0 (pid 3642) produces s
```

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



```
                                                Consumer 0 (pid 3645) consumes l
Producer 1 (pid 3643) produces m
                                                Consumer 2 (pid 3647) consumes m
Producer 0 (pid 3642) produces n
                                                Consumer 1 (pid 3646) consumes n
Producer 2 (pid 3644) produces o
                                                Consumer 0 (pid 3645) consumes o
Producer 0 (pid 3642) produces p
                                                Consumer 2 (pid 3647) consumes p
Producer 1 (pid 3643) produces q
                                                Consumer 1 (pid 3646) consumes q
Producer 2 (pid 3644) produces r
                                                Consumer 0 (pid 3645) consumes r
Producer 0 (pid 3642) produces s
                                                Consumer 1 (pid 3646) consumes s
Producer 1 (pid 3643) produces t
                                                Consumer 2 (pid 3647) consumes t
Producer 2 (pid 3644) produces u
                                                Consumer 0 (pid 3645) consumes u
Producer 0 (pid 3642) produces v
Producer 1 (pid 3643) produces w
                                                Consumer 2 (pid 3647) consumes w
                                                Consumer 1 (pid 3646) consumes v
Producer 0 (pid 3642) produces x
Producer 1 (pid 3643) produces y
                                                Consumer 0 (pid 3645) consumes y
                                                Consumer 2 (pid 3647) consumes x
Producer 2 (pid 3644) produces z
                                                Consumer 1 (pid 3646) consumes z
Process 3645 returns 0
Process 3646 returns 0
Process 3644 returns 0
Process 3647 returns 0
Process 3642 returns 0
Process 3643 returns 0
daniil@daniil-VirtualBox:~/Desktop/operating-system/sem_5/lab05$ █
```

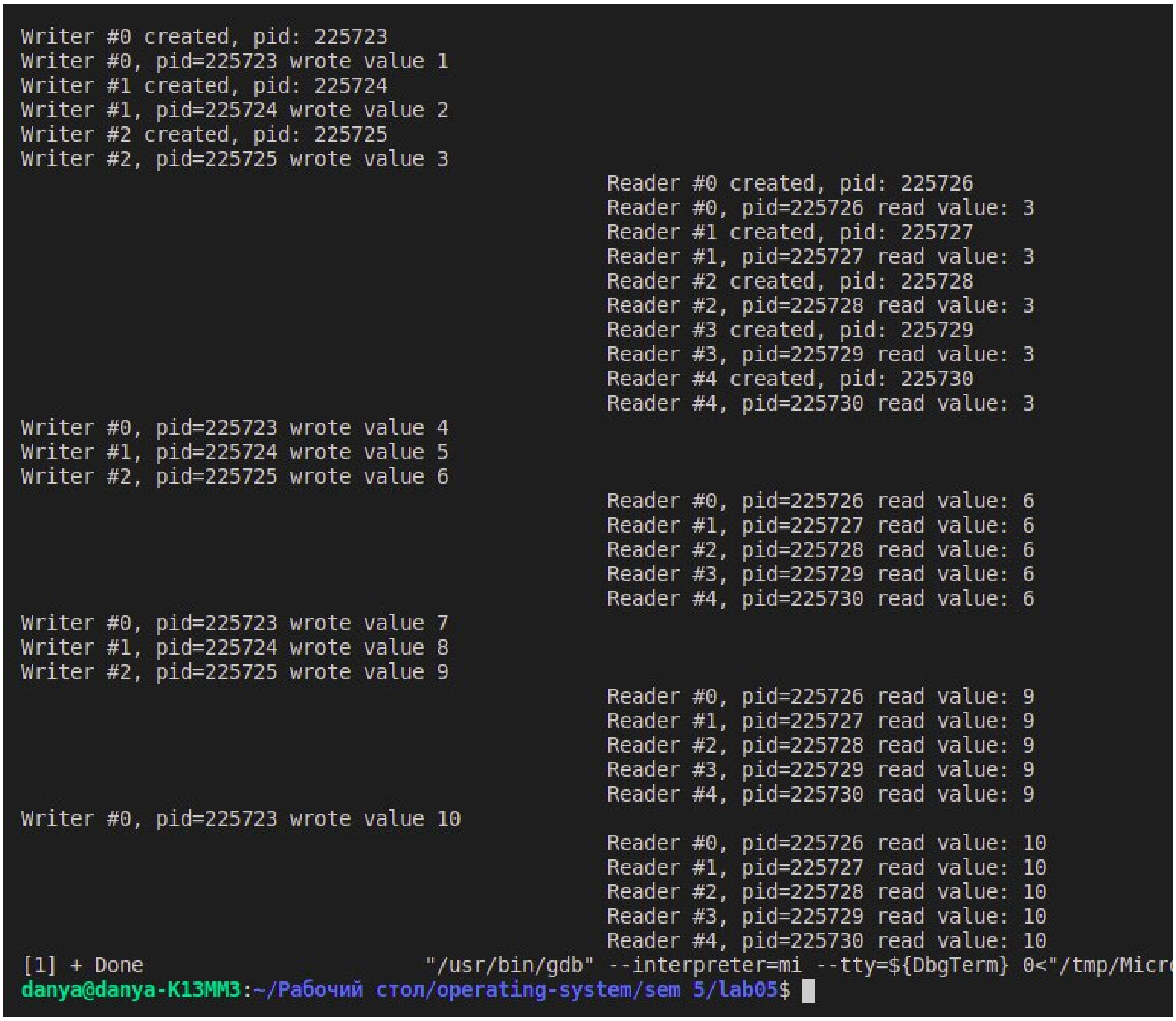
Рис. 2: Пример работы программы. BUFFER_SIZE = 3 (конец вывода)

Листинг 2: Задача читатели-писатели

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

#define READER_BORDER "\t\t\t\t\t\t"

#define PERMS S_IRWXU | S_IRWXG | S_IRWXO

#define MAX_VALUE 10
#define WRITERS 3
#define READERS 5

#define ACTIVE_WR 0
#define ACTIVE_RR 1
#define WAITING_W 2
#define WAITING_R 3

#define V 1
#define P -1
#define Z 0

#define SEM_ACTIVE_WR 0
#define SEM_ACTIVE_RR 1
#define SEM_WAITING_W 2
#define SEM_WAITING_R 3

struct sembuf start_reading[] = {
  {SEM_WAITING_R, V, SEM_UNDO},
  {SEM_ACTIVE_WR, Z, SEM_UNDO},
  {SEM_WAITING_W, Z, SEM_UNDO},
  {SEM_ACTIVE_RR, V, SEM_UNDO},
  {SEM_WAITING_R, P, SEM_UNDO}
};
struct sembuf stop_read[] = {
  {SEM_ACTIVE_RR, P, SEM_UNDO}
};

struct sembuf start_writing[] = {
  {SEM_WAITING_W, V, SEM_UNDO},
  {SEM_ACTIVE_RR, Z, SEM_UNDO},
  {SEM_ACTIVE_WR, Z, SEM_UNDO},
  {SEM_ACTIVE_WR, V, SEM_UNDO},
```

```c
  {SEM_WAITING_W, P, SEM_UNDO}
};
struct sembuf stop_write[] = {
  {SEM_ACTIVE_WR, P, SEM_UNDO}
};

int *shared_value;

pid_t *child_pids;


void kill_writers() {
  for (int i = 0; i < WRITERS; i++) {
    if (child_pids[i] == getpid()) {
      continue;
    }
    kill(child_pids[i], SIGTERM);
  }
}

int init_sh_mem() {
  int fd = shmget(IPC_PRIVATE, sizeof(int), IPC_CREAT | PERMS);
  if (fd == -1) {
    perror("shmget");
    exit(1);
  }
  return fd;
}

int *get_sh_mem_addr(const int fd) {
  int *addr = (int *) shmat(fd, NULL, 0);
  if (addr == (int *) -1) {
    perror("shmat error");
    exit(1);
  }
  return addr;
}

int init_semaphores() {
  int semid = semget(IPC_PRIVATE, 4, IPC_CREAT | PERMS);
  if (semid == -1) {
    perror("semget error");
    exit(1);
  }

  int mem_ctrl    = semctl(semid, WAITING_R, SETVAL, 0);
  int writer_ctrl = semctl(semid, ACTIVE_WR, SETVAL, 0);
  int reader_ctrl = semctl(semid, ACTIVE_RR, SETVAL, 0);
  int wait_ctrl   = semctl(semid, WAITING_W, SETVAL, 0);

  if (mem_ctrl == -1 || writer_ctrl == -1 || reader_ctrl == -1 || wait_ctrl == -1) {
    perror("semctl");
    exit(1);
  }

  return semid;
}

void writer(int semid, int number) {
  int can = semop(semid, start_writing, 5);
  if (can == -1) {
    perror("semop start writing error");
    exit(1);
  }
  // Z condition for stopping all writers
  if (*shared_value >= MAX_VALUE) {
    kill_writers();
    int sem_op_stop = semop(semid, stop_write, 1);
    if (sem_op_stop == -1) {
      perror("semop stor write error");
      exit(1);
    }
    exit(0);
  }
  // write
  (*shared_value)++;
  printf("Writer #%d, pid=%d wrote value %d\n", number, getpid(), *shared_value);

  int sem_op_stop = semop(semid, stop_write, 1);
  if (sem_op_stop == -1) {
    perror("semop stop_write");
    exit(1);
  }

  sleep(rand() % 10);
}

void reader(int semid, int number) {
  int can = semop(semid, start_reading, 5);
  if (can == -1) {
    perror("semop start_reading");
    exit(1);
  }
  // read
  int val = *shared_value;
  printf(READER_BORDER"Reader #%d, pid=%d read value: %d\n", number,
     getpid(), val);

  int sem_op_stop = semop(semid, stop_read, 1);
  if (sem_op_stop == -1) {
    perror("semop stop_read");
    exit(1);
  }
  // Z condition for stopping all readers
  if (val >= MAX_VALUE) {
    exit(0);
  }

  sleep(rand() % 10);
}

void init_writer(int number, int sem_id) {
  pid_t pid;

  if ((pid = fork()) == -1) {
    printf("Can't fork");
    exit(1);
  }

  if (pid == 0) {
    printf("Writer #%d created, pid: %d\n", number, getpid());
    while (1) {
      writer(sem_id, number);
    }
  }
  else {
    child_pids[number] = pid;
  }
}

void init_reader(int number, const int sem_id) {
  pid_t pid;

  if ((pid = fork()) == -1) {
    printf("Can't fork");
    exit(1);
  }

  if (pid == 0) {
    printf(READER_BORDER"Reader #%d created, pid: %d\n", number, getpid↩
       ());
    while (1) {
      reader(sem_id, number);
    }
  }
```

```c
  else {
    child_pids[WRITERS + number] = pid;
  }
}

int main() {
  int sh_mem_fd = init_sh_mem();
  int *sh_mem = get_sh_mem_addr(sh_mem_fd);

  shared_value = sh_mem;
  *shared_value = 0;
  child_pids = shared_value + 1;

  int semid = init_semaphores();

  for (int i = 0; i < WRITERS; i++) {
    init_writer(i, semid);
  }

  for (int i = 0; i < READERS; i++) {
    init_reader(i, semid);
  }

  for (int i = 0; i < WRITERS + READERS; i++) {
    int *status;
    wait(status);
  }
}
```



```
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                       "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm} 0<"/tmp/Micr
danya@danya-K13MM3:~/Рабочий стол/operating-system/sem 5/lab05$ 
```

Рис. 3: Пример работы программы.