

МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Балтийский государственный технический университет «ВОЕНМЕХ» им. Д.Ф. Устинова»
(БГТУ «ВОЕНМЕХ» им. Д.Ф. Устинова)

Кафедра	<u>О7</u>	<u>Информационные системы и программная инженерия</u>
	шифр	наименование кафедры, по которой выполняется работа
Дисциплина		<u>Информационные технологии</u>
		наименование дисциплины

ПРАКТИЧЕСКАЯ РАБОТА 7

номер (при наличии)

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ СИ В ОС
LINUX

Вариант 9

при наличии указать тему лабораторной работы и (или) номер варианта

ОБУЧАЮЩИЙСЯ

группы О726Б

Махов Н.М.

подпись

фамилия и инициалы

дата сдачи

ПРОВЕРИЛ

ученая степень, ученое звание, должность

Мажайцев Е.А.

подпись

фамилия и инициалы

Оценка / балльная оценка

дата проверки

Санкт-Петербург
20 24 г.

Текст задания:

Вариант 9

Написать две программы: «сервер» и «клиент», взаимодействующие через разделяемую память: «клиент» отправляет текстовые строки, а «сервер» их выводит на экран. Для синхронизации работы программ и работы с разделяемой памятью использовать семафоры.

Текст программы:

pair.h

```
/* Файл sem_defs.h */
/* Ключ разделяемой памяти */
#define SEG_KEY 0x12345678
/* Ключ набора семафоров */
#define SEM_KEY 0x12345678

/* Максимальная длина передаваемой строки */
#define MAXSTRLEN 100
/* union для вызова semctl() */
union semun {
    int val;
};
```

server.c (main)

```
/* Файл sem_main.c - программа-"сервер" */
#include <stdio.h>
#include <unistd.h>
#include <sys/shm.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <string.h>
#include "pair.h"
int main()
{
    int shmid; /* ID сегмента разделяемой памяти */
    char *str; /* Указатель на принимаемую строку */
    int semid; /* ID набора семафоров */
    int retcode = 1; /* Код возврата из программы */
    union semun semcd; /* Используется semctl() */
    struct sembuf semoper[1]; /* Используется semop() */
    int exitflag=1; /* Флаг выхода из цикла приема строк */
    /* Создание сегмента разделяемой памяти */
    if ( (shmid = shmget(SEG_KEY, MAXSTRLEN+1,
                        IPC_CREAT | IPC_EXCL | 0600)) < 0) {
        printf("Ошибка shmget()!\n"); return 1;
    }
    /* Подключение сегмента разделяемой памяти */
    if((str=(char *)shmat(shmid,NULL,0))==(char *)(-1)) {
        printf("shmat() error\n"); goto destroy_segment;
    }
    /* Создание набора семафоров */
    if ( (semid = semget(SEM_KEY, 1,
```

```

                                IPC_CREAT | IPC_EXCL | 0600)) < 0 ) {
    printf("Ошибка semget()!\n");
    goto destroy_segment;
}
/* Установка начального значения семафора, равного 1 */
semcd.val = 1;
if (semctl(semid, 0, SETVAL, semcd) < 0) {
    printf("Ошибка semctl()!\n");

    goto destroy_semaphore;
}
/* Цикл приема строк через разделяемую память */
while(exitflag) {
    /* Обеспечиваем монопольный доступ к ресурсу
    ( разделяемому сегменту памяти ) */
    /* P-операция */
    semoper[0].sem_num = 0; /* Номер семафора в наборе */
    semoper[0].sem_op = -1; /* Операция */
    semoper[0].sem_flg = 0;
    semop(semid, semoper, 1); /* Выполнение операции */
    if (str[0]) { /* Если строка имеется ... */
        printf("Принята строка: %s\n", str);
        /* Проверка: передана команда выхода? */
        if (strncmp(str, "EXIT", 4) == 0) {
            exitflag = 0;
            printf("ВЫХОД.\n");
        }
        /* Сбрасываем признак наличия строки */
        str[0] = 0;
    }
    /* Освобождаем ресурс */
    /* V-операция */
    semoper[0].sem_num = 0; /* Номер семафора в наборе */
    semoper[0].sem_op = 1; /* Операция */
    semoper[0].sem_flg = 0;
    semop(semid, semoper, 1); /* Выполнение операции */
} /* Конец цикла приема строк */
/* Нормальное завершение */
/* Отсоединение сегмента */
shmdt(str);
retcode = 0;
destroy_semaphore:
    /* Уничтожение набора семафоров */
    semctl(semid, IPC_RMID, 0);
destroy_segment:
    /* Уничтожение сегмента разделяемой памяти */
    shmctl(shmid, IPC_RMID, NULL);
    return retcode;
}

```

client.c

```

/* Файл sem_send.c - программа-клиент */
#include <stdio.h>
#include <unistd.h>
#include <sys/shm.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include "pair.h"
int main()

```

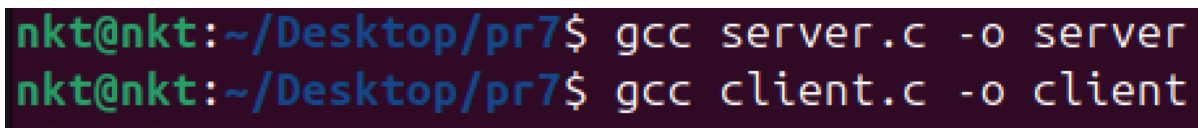
```

{
    int shmid; /* ID сегмента разделяемой памяти */
    char *str; /* Передаваемая строка */
    int semid; /* ID набора семафоров */
    struct sembuf semoper[1]; /* Используется semop() */
    /* Получение доступа к разделяемой памяти */
    if ( (shmid = shmget(SEG_KEY, MAXSTRLEN+1, 0600)) < 0 ) {
        printf("Ошибка shmget()!\n"); return 1;
    }
    /* Подключение сегмента разделяемой памяти */
    if ((str=(char *)shmat(shmid,NULL,0))==(char *) (-1)) {
        printf("shmat() error\n"); return 1;
    }
    /* Получение доступа к набору семафоров */
    if ( (semid = semget(SEM_KEY, 1, 0600)) < 0 ) {
        printf("Ошибка semget()!\n");
        return 2;
    }
    /* Получение монопольного доступа к разделяемой памяти */
    /* P-операция */
    semoper[0].sem_num = 0; /* Номер семафора в наборе */
    semoper[0].sem_op = -1; /* операция */
    semoper[0].sem_flg = 0;
    semop(semid, semoper, 1); /* Выполнение операции */
    /* Ввод строки с клавиатуры */
    printf("Введите строку: ");
    fgets(str, MAXSTRLEN, stdin);
    /* Освобождаем ресурс (разделяемую память) */
    /* V-операция */
    semoper[0].sem_num = 0; /* Номер семафора в наборе */
    semoper[0].sem_op = 1; /* операция */
    semoper[0].sem_flg = 0;
    semop(semid, semoper, 1); /* Выполнить операцию */
    /* Отключение сегмента разделяемой памяти */
    shmdt(str);
    return 0;
}

```

Результат работы:

Компилируем файлы server.c и client.c, результат представлен на рисунке 1. Сервер - main.



```

nkt@nkt:~/Desktop/pr7$ gcc server.c -o server
nkt@nkt:~/Desktop/pr7$ gcc client.c -o client

```

Рисунок 1 — Компиляция файлов сервера и клиента.

После компиляции, директория для выполнения работы, имеет вид как на рисунке 2.

```

-rwxrwxr-x 1 nkt nkt 70744 Sep 16 19:48 client
-rw-rw-r-- 1 nkt nkt 2057 Sep 16 19:48 client.c
-rw-rw-r-- 1 nkt nkt 355 Sep 16 19:41 pair.h
-rwxrwxr-x 1 nkt nkt 70792 Sep 16 19:48 server
-rw-rw-r-- 1 nkt nkt 3622 Sep 16 19:47 server.c

```

Рисунок 2 — Директория со скомпилированными файлами

Запускаем сервер в фоновом режиме (&), номер процесса 4027, с помощью команды `ipcs` можно убедиться в наличии сегмента разделяемой памяти и набора семафоров с ключами `0x12345678`, рисунок 3.

```

nkt@nkt:~/Desktop/pr7$ ./server &
[1] 4027
nkt@nkt:~/Desktop/pr7$ ipcs

----- Message Queues -----
key          msqid          owner          perms          used-bytes   messages

----- Shared Memory Segments -----
key          shmid          owner          perms          bytes        nattch       status
0x12345678  0              nkt            600            101          1

----- Semaphore Arrays -----
key          semid          owner          perms          nsems
0x12345678  1              nkt            600            1

```

Рисунок 3 — Наличие сегмента памяти и семафоры

Запускаем клиент, пользователю предлагается ввести строку, вывод, вводим, результат на рисунке 4.

```

nkt@nkt:~/Desktop/pr7$ ./client
Введите строку: test string
Принята строка: test string

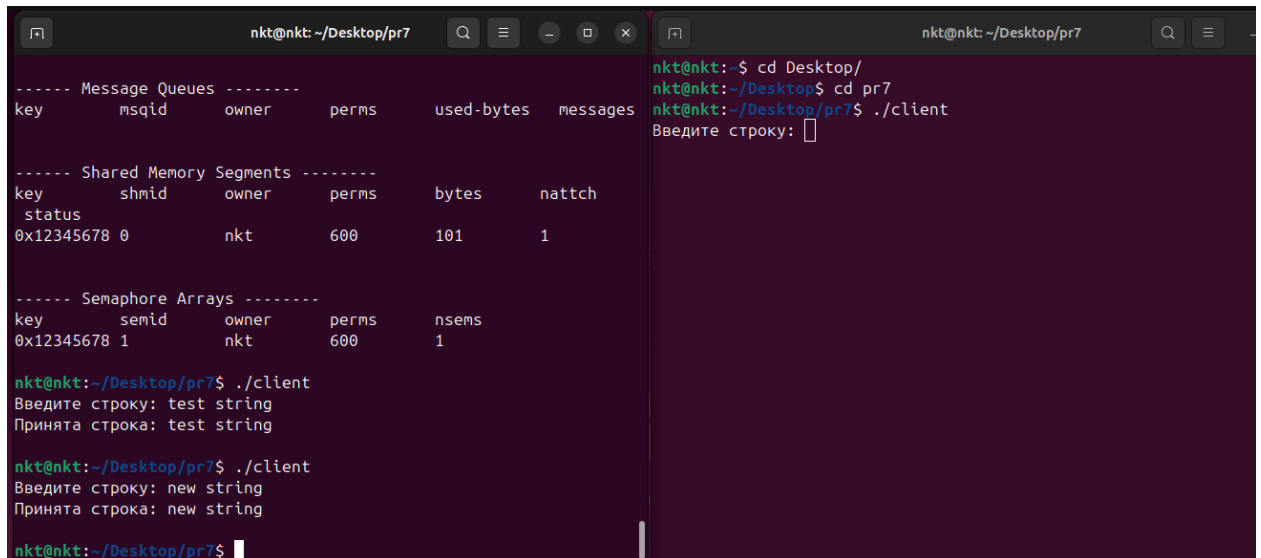
```

Рисунок 4 — Ввод строки, вывод через сервер.

Запустим клиента повторно, пользователю предлагают ввести строку, не вводим, открываем второе окно в терминале, запускаем клиента. Программа просто запускается, сервер не предлагает ввести строку, потому что семафор

=0, вводим строку в первом окне, теперь на втором можно ввести строку.

Результат представлен на рисунке 5.



```
nkt@nkt: ~/Desktop/pr7
----- Message Queues -----
key      msqid    owner    perms    used-bytes   messages
----- Shared Memory Segments -----
key      shmid    owner    perms    bytes       nattch
status
0x12345678 0      nkt      600     101         1

----- Semaphore Arrays -----
key      semid     owner    perms    nsems
0x12345678 1      nkt      600     1

nkt@nkt:~/Desktop/pr7$ ./client
Введите строку: test string
Принята строка: test string

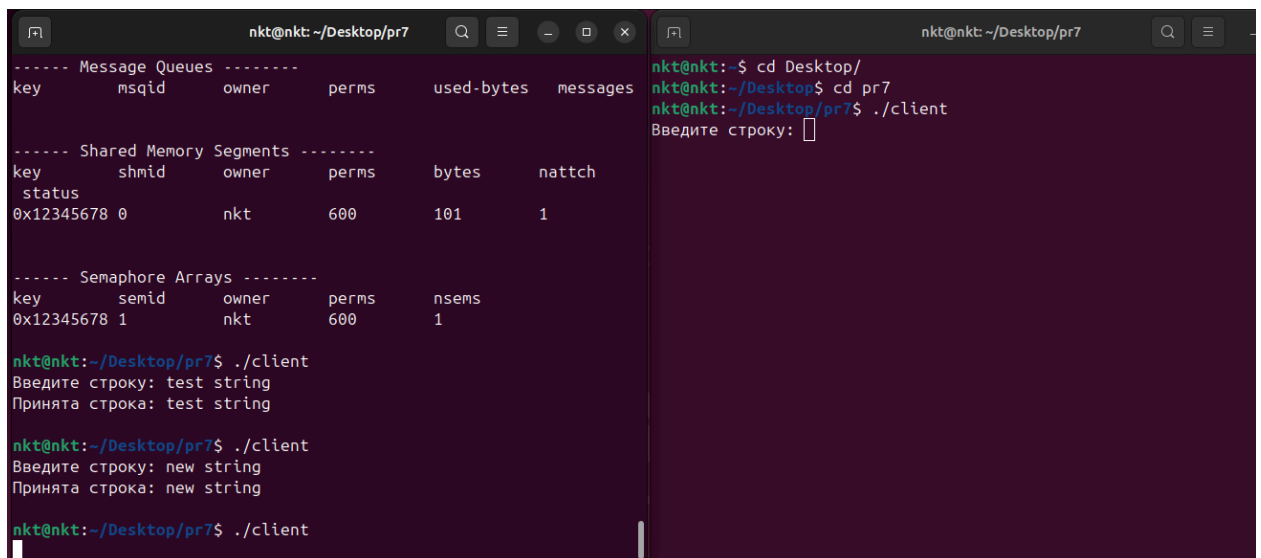
nkt@nkt:~/Desktop/pr7$ ./client
Введите строку: new string
Принята строка: new string

nkt@nkt:~/Desktop/pr7$
```

```
nkt@nkt:~/Desktop/pr7
nkt@nkt:~$ cd Desktop/
nkt@nkt:~/Desktop$ cd pr7
nkt@nkt:~/Desktop/pr7$ ./client
Введите строку: 
```

Рисунок 5 — Запуск client на двух окнах терминала, сервер обрабатывает только левое окно

Пример запуска клиента на втором и первом окне, пока сервер не обработает второе окно, первое в режиме ожидания. Результат на рисунке 6.



```
nkt@nkt: ~/Desktop/pr7
----- Message Queues -----
key      msqid    owner    perms    used-bytes   messages
----- Shared Memory Segments -----
key      shmid    owner    perms    bytes       nattch
status
0x12345678 0      nkt      600     101         1

----- Semaphore Arrays -----
key      semid     owner    perms    nsems
0x12345678 1      nkt      600     1

nkt@nkt:~/Desktop/pr7$ ./client
Введите строку: test string
Принята строка: test string

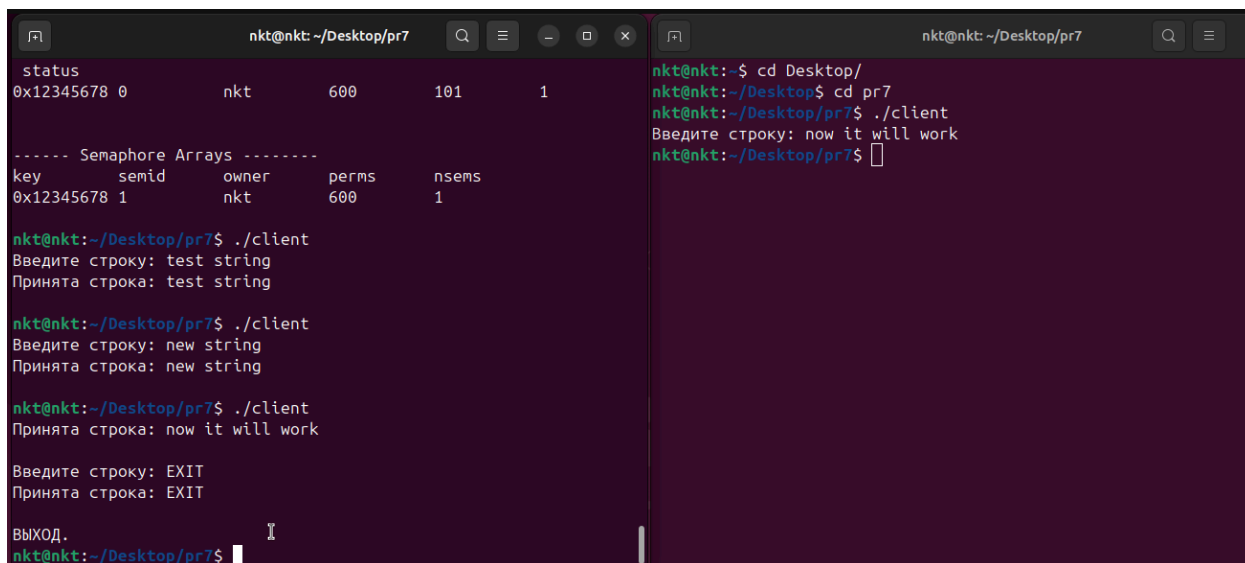
nkt@nkt:~/Desktop/pr7$ ./client
Введите строку: new string
Принята строка: new string

nkt@nkt:~/Desktop/pr7$ ./client
```

```
nkt@nkt:~/Desktop/pr7
nkt@nkt:~$ cd Desktop/
nkt@nkt:~/Desktop$ cd pr7
nkt@nkt:~/Desktop/pr7$ ./client
Введите строку: 
```

Рисунок 6 — Запуск client на двух окнах терминала, сервер обрабатывает только правое окно

Завершение работы, EXIT, результат на рисунке 7.



The image shows two terminal windows side-by-side. The left window displays the output of the 'status' command, showing semaphore details for key 0x12345678. It then shows three successful client interactions with the semaphore, each receiving a string. The right window shows the user navigating to the 'pr7' directory and running './client', which prompts for a string and receives 'now it will work'.

```
nkt@nkt: ~/Desktop/pr7
status
0x12345678 0      nkt      600      101      1

----- Semaphore Arrays -----
key      semid      owner      perms      nsems
0x12345678 1      nkt      600      1

nkt@nkt:~/Desktop/pr7$ ./client
Введите строку: test string
Принята строка: test string

nkt@nkt:~/Desktop/pr7$ ./client
Введите строку: new string
Принята строка: new string

nkt@nkt:~/Desktop/pr7$ ./client
Введите строку: now it will work
Принята строка: now it will work

Введите строку: EXIT
Принята строка: EXIT

Выход.
nkt@nkt:~/Desktop/pr7$
```

```
nkt@nkt:~$ cd Desktop/
nkt@nkt:~/Desktop$ cd pr7
nkt@nkt:~/Desktop/pr7$ ./client
Введите строку: now it will work
nkt@nkt:~/Desktop/pr7$
```

Рисунок 7 — Завершение работы

Вывод:

Семафоры в Linux помогают синхронизировать потоки и предотвращают ошибки. Разделение процессов улучшает использование ресурсов и безопасность. Эффективное управление памятью оптимизирует работу приложений и предотвращает утечки. Все эти механизмы делают программы более стабильными и безопасными в многозадачной среде.