

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ № 14

дисциплина: Операционные системы

Студенты: Подмогильный Иван Александрович

Бешкуров Михаил Борисович

Группа: НКНбд-01-18_

МОСКВА

2019 г.

ЦЕЛЬ РАБОТЫ

Приобретение практических навыков работы с очередями сообщений.

ОПИСАНИЕ РЕЗУЛЬТАТОВ ВЫПОЛНЕНИЯ ЗАДАНИЙ

Изучил приведённые в тексте программы *server.c* и *client.c*

```
all: server client
server: server.c common.h
    gcc server.c -o server
client: client.c common.h
    gcc client.c -o client
clean:
    -rm server client *.o
```

```
/*
 * common.h - заголовочный файл со стандартными определениями
 */
#ifndef __COMMON_H__
#define __COMMON_H__
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#define MAXBUF 80
#define PERM 0666
/*
 * Определение структуры сообщения. Первым элементом структуры
 * должен быть элемент типа long, указывающий на тип сообщения.
 * Другие элементы могут быть определены дополнительно.
 */
typedef struct my_msgbuf
{
    long mtype;
    char buff[MAXBUF];
} my_message_t;

#endif /* __COMMON_H__ */
```

```

/home/ivan/server.c  [---]  2 L:[ 1+ 0  1/ 72] *(2  /1679b) 0010 0x00A  [*][X]
/*
 * server.c - реализация сервера
 *
 * чтобы запустить пример необходимо:
 * 1. запустить программу server на одной консоли;
 * 2. запустить программу client на другой консоли.
 */
#include "common.h"

int
main()
{
    my_message_t message;
    key_t key;
    int msgid;
    int length;
    int n;

    /* баннер */
    printf("Message Queue Server...\n");

    /* получить ключ */
    if ((key = ftok("server", 'A')) < 0)
    {
        fprintf(stderr,
            "%s: Невозможно получить ключ (%s)\n",
            __FILE__, strerror(errno));
        exit(-1);
    }

    /* создать очередь сообщений */
    if ((msgid = msgget(key, PERM | IPC_CREAT)) < 0)
    {
        fprintf(stderr,
            "%s: Невозможно создать очередь (%s) \n",
            __FILE__, strerror(errno));
        exit(-1);
    }

    /* прочитать сообщение */
    message.mtype = 1L;
    n = msgrcv(msgid, &message, sizeof (message), message.mtype, 0);

    /* если сообщение поступило, напечатать содержимое */
    if (n <= 0)
    {
        fprintf(stderr,
            "%s: Ошибка чтения сообщения (%s) \n",
            __FILE__, strerror(errno));
        exit(-1);
    }
    else
    {
        if (write(1, message.buf, n) != n)
        {
            fprintf(stderr,
                "%s: Ошибка вывода (%s) \n",
                __FILE__, strerror(errno));
            exit(-2);
        }
    }

    /* удалить очередь сообщений */
    if (msgctl(msgid, IPC_RMID, 0) < 0)
    {
        fprintf(stderr,
            "%s: Ошибка удаления очереди (%s)\n",
            __FILE__, strerror(errno));
        exit(-3);
    }
    exit(0);
}

```

1Помощь 2Сохранить 3Блок 4Замена 5Копия 6Пере-тить 7Поиск 8Удалить 9МенюМС 10Выход

```
/home/ivan/client.c [----] 6 L:[ 1+ 5 6/ 63] *(210 /1534b) 1079 0x437 [*][X]
/*
 * client.c - реализация сервера
 *
 * чтобы запустить пример необходимо:
 * 1. запустить программу server на одной консоли;
 * 2. запустить программу client на другой консоли.
 */
#include "common.h"
#define MESSAGE "Hello world!!!\n"

int
main()
{
    my_message_t message;
    key_t key;
    int msgid;
    int length;
    int n;

    /* баннер */
    printf("Message Queue Client...\n");

    /* получить ключ */
    if((key=ftok("server",'A'))<0)
    {
        fprintf(stderr,
            "%s: Невозможно получить ключ (%s)\n",
            __FILE__, strerror(errno));
        exit(-1);
    }

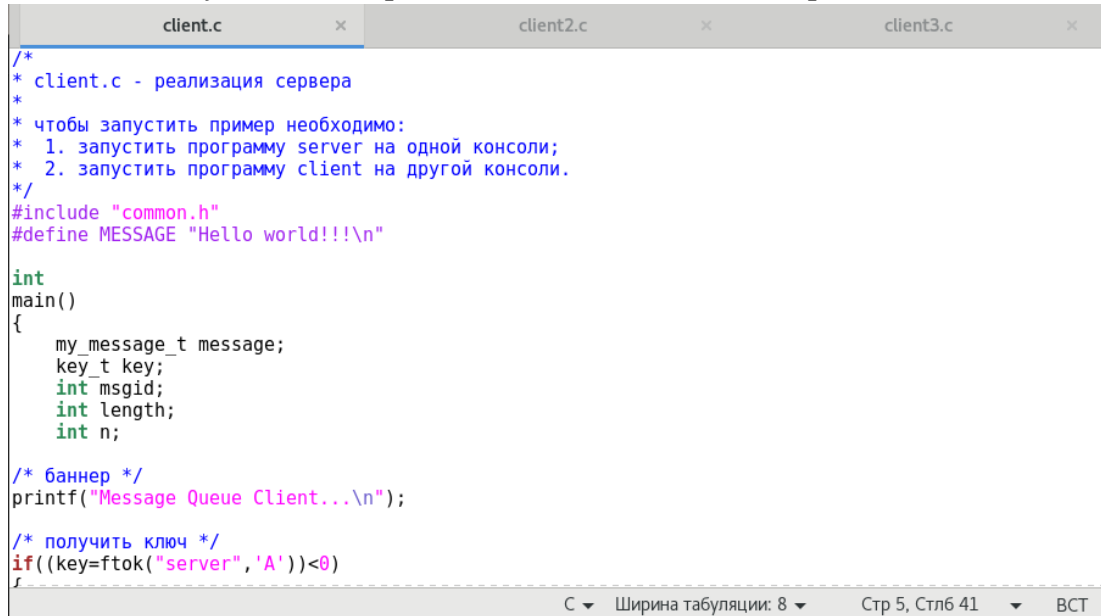
    /* получить доступ к очереди сообщений, очередь уже должна
     * быть создана сервером
     */
    if((msgid=msgget(key,0))<0)
    {
        fprintf(stderr,
            "%s: Невозможно получить доступ к очереди (%s)\n",
            __FILE__, strerror(errno));
        exit(-2);
    }

    /*подготовка сообщения*/
    message.mtype = 1L;
    if((length=sprintf(message.buff, MESSAGE))<0)
    {
        fprintf(stderr,
            "%s: Ошибка копирования в буфер (%s)\n",
            __FILE__, strerror(errno));exit(-3);
    }

    /* передача сообщения */
    if(msgsnd(msgid, (void*)&message, length,0)<0)
    {
        fprintf(stderr,
            "%s: Ошибка записи сообщения в очередь (%s)\n",
            __FILE__, strerror(errno));
        exit(-4);
    }
    exit(0);
}
```

Взяв данные примеры за образец, написал аналогичные программы, внося следующие изменения:

1. Работает 1 сервер и несколько клиентов (3 клиента с одинаковыми скриптами).
2. Мультиплексировал сообщения в очередь. Каждый клиент



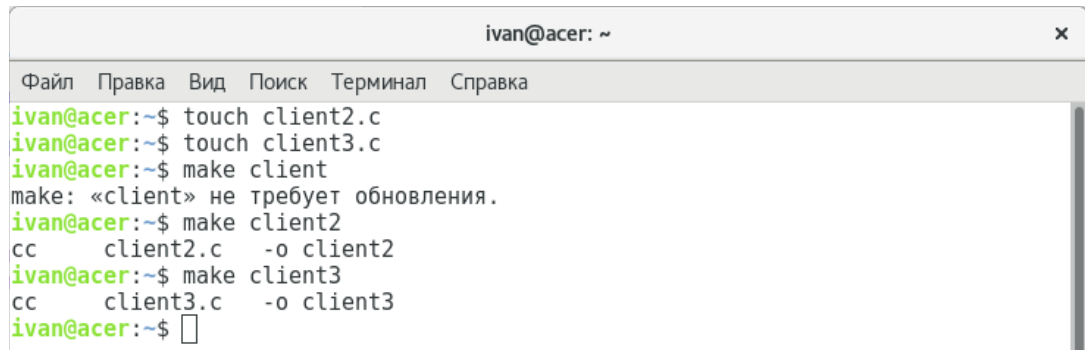
```
client.c
client2.c
client3.c

/*
 * client.c - реализация сервера
 *
 * чтобы запустить пример необходимо:
 * 1. запустить программу server на одной консоли;
 * 2. запустить программу client на другой консоли.
 */
#include "common.h"
#define MESSAGE "Hello world!!!\n"

int
main()
{
    my_message_t message;
    key_t key;
    int msgid;
    int length;
    int n;

    /* баннер */
    printf("Message Queue Client...\n");

    /* получить ключ */
    if((key=ftok("server", 'A'))<0)
```



```
ivan@acer: ~
Файл  Правка  Вид  Поиск  Терминал  Справка
ivan@acer:~$ touch client2.c
ivan@acer:~$ touch client3.c
ivan@acer:~$ make client
make: «client» не требует обновления.
ivan@acer:~$ make client2
cc      client2.c  -o client2
ivan@acer:~$ make client3
cc      client3.c  -o client3
ivan@acer:~$
```

посылает свой тип сообщений (mtype равен PID процесса) и сервер распечатывает их сообщения, указывая для каждого сообщения от какого клиента оно пришло.

Открыть client3.c Сохранить

```
/*
 * client.c - реализация сервера
 *
 * чтобы запустить пример необходимо:
 */
/*
 * server.c - реализация сервера
 *
 * чтобы запустить пример необходимо:
 * 1. запустить программу server на одной консоли;
 * 2. запустить программу client на другой консоли.
 */
#include "common.h"

int
main()
{
    my_message_t message;
    key_t key;
    int msgid;
    int length;
    int n;
    int PID = getpid();
    /* баннер */
    printf("Message Queue Server...\n");

    /* получить ключ */
    if ((key = ftok("server", 'A')) < 0)
    {
        fprintf(stderr,
            "%s: Невозможно получить ключ (%s)\n",
            __FILE__, strerror(errno));
        exit(-1);
    }

    /* создать очередь сообщений */
    if ((msgid = msgget(key, PERM | IPC_CREAT)) < 0)
    {
        fprintf(stderr,
            "%s: Невозможно создать очередь (%s) \n",
            __FILE__, strerror(errno));
        exit(-1);
    }

    /* создать очередь сообщений */
    if ((msgid = msgget(key, PERM | IPC_CREAT)) < 0)
    {
        fprintf(stderr,
            "%s: Невозможно создать очередь (%s) \n",
            __FILE__, strerror(errno));
        exit(-1);
    }

    while(1){
        /* прочитать сообщение */
        message.mtype = 1L;
        n = msgrcv(msgid, &message, sizeof (message), message.mtype, 0);

        /* если сообщение поступило, напечатать содержимое */
        /*if (n<=0)
        {
            fprintf(stderr,
                "%s: Ошибка чтения сообщения (%s) \n",
                __FILE__, strerror(errno));
            exit(-1);
        }*/
        if(n>0)
        {
            if (write(1, message.buf, n) != n)
            {
                fprintf(stderr,
                    "%s: Ошибка вывода (%s) \n",
                    __FILE__, strerror(errno));
                exit(-2);
            }
        }
    }

    /* удалить очередь сообщений */
    if (msgctl(msgid, IPC_RMID, 0)<0)
    {
        fprintf(stderr,
            "%s: Ошибка удаления очереди (%s)\n",
            __FILE__, strerror(errno));
        exit(-3);
    }

    exit(0);
}
```

```
Обзор Терминал
Субота, 02:32
ivan@acer: ~
Файл Правка Вид Поиск Терминал Справка
client.c: In function 'main':
client.c:20:13: warning: implicit declaration of function 'getpid' [-Wimplicit-function-declaration]
pid_t PID = getpid();
server.c: In function 'main':
server.c:18:15: warning: implicit declaration of function 'write' [-Wimplicit-function-declaration]
if (write(1, message.buff, n) != n)
ivan@acer:~$ make server
gcc server.c -o server
ivan@acer:~$ ./server
Message Queue Server...
Hello world!!! 1788
Hello world!!!
Hello world!!!
Hello world!!! 1867
Hello world!!! 1882
Hello world!!! 1884
Hello world!!! 1885
ivan@acer:~$
ivan@acer: ~
Файл Правка Вид Поиск Терминал Справка
client.c: In function 'main':
client.c:20:13: warning: implicit declaration of function 'getpid' [-Wimplicit-function-declaration]
pid_t PID = getpid();
ivan@acer:~$ ./client2
Message Queue Client...
ivan@acer:~$
ivan@acer: ~
Файл Правка Вид Поиск Терминал Справка
client2.c: In function 'main':
client2.c:20:13: warning: implicit declaration of function 'getpid' [-Wimplicit-function-declaration]
pid_t PID = getpid();
ivan@acer:~$ make client2
make: «client2» не требует обновления.
ivan@acer:~$ ./client2
Message Queue Client...
ivan@acer:~$
```

ВЫВОД

Приобрел практические навыки работы с очередями сообщений.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Очереди сообщений входят в состав средств System V IPC. Пространством имен очередей сообщений является то же самое множество значений ключа, которые генерируются с помощью функции `ftok()`. Эта функция действует в предположении, что для конкретного приложения, использующего IPC, клиент и сервер используют одно и то же полное имя объекта IPC, имеющее какое-то значение в контексте приложения. Это может быть имя демона сервера или имя файла данных, используемого сервером, или имя еще какого-нибудь объекта файловой системы. Если клиенту и серверу для связи требуется только один канал IPC, идентификатору можно присвоить, например, значение 1. Если требуется несколько каналов IPC (например, один от сервера к клиенту и один в обратную сторону), идентификаторы должны иметь разные значения: например, 1 и 2. После того как клиент и сервер договорятся о полном имени и идентификаторе, они оба вызывают функцию `ftok` для получения одинакового ключа IPC.

2. Ключ генерируется на основе имени файла (берётся его i-node).

Для генерации нескольких ключей на основе одного имени файла используется дополнительный идентификатор проекта (обычно ASCII-символ). Можно, так как это бинарный файл. На основе имени этого файла и будет генерироваться ключ.

3. Дескриптор файла - это целое число без знака, с помощью которого процесс обращается к открытому файлу. Количество дескрипторов файлов, доступных процессу, ограничено параметром `/OPEN_MAX`, заданным в файле `sys/limits.h`.

Межпроцессное взаимодействие (Inter-process communication (IPC)) — это набор методов для обмена данными между потоками процессов. Процессы могут быть запущены как на одном и том же компьютере, так и на разных, соединённых сетью. IPC бывают нескольких типов: «сигнал», «сокет», «семафор», «файл», «сообщение»...

4. Системные вызовы создания очереди, записи сообщения в очередь, извлечения сообщения из очереди и управления очередью:

- `msgget` служит для образования новой очереди сообщений или получения дескриптора существующей очереди
- `msgsnd` служит для отправки сообщения (его постановки в очередь сообщений)
- `msgrcv` служит для приёма сообщения (выборки сообщения из очереди)
- `msgctl` служит для выполнения управляющих действий.

После открытия очереди сообщений с помощью функции `msgget` можно помещать сообщения в эту очередь с помощью `msgsnd`. Ид сообщения должен быть больше нуля, поскольку неположительные типы используются в качестве специальной команды функции `msgrcv`