# 08. Межпроцессное взаимодействие (продолжение)

Егор Орлов

Курс: UNIX-DEV-SYS. Системное программирование в среде UNIX (Linux/FreeBSD). ВИШ СПбПУ, 2021

## Содержание



# 1. Разделяемая память POSIX

## 1.1. Идея

- Область памяти POSIX определяется идентификатором (именем)
- Приложение может создать сегмент памяти с запрошенным именем, либо подключиться по имени к уже существующему
- В linux имена объектов разделяемой памяти отображаются в псевдофайловую систему /dev/shm
- Внутри приложения области памяти идентифицируются дескрипторами (такими же как файловые)
- Сами функции для работы с разделяемой памятью POSIX похожи на функции низкоуровневого ввода-вывода

## 1.2. Реализация

- Берем MMF, которая уже по сути общая память для процессов, но связанная с каким-то файлом на диске
- Связываем MMF не с реальным файлом, а с файлом на псевдофайловой системе, специально создавая его там
- Чтобы определить размер будуще области памяти, мы устанавливаем размер этого псевдофайла
- Вызываем mmap на него и имеем по итогам указатель на область памяти, с которым и работаем.

## 1.3. Функции для работы с памятью

```
#include <sys/mman.h>
int shm_open(const char *name, int oflag, mode_t mode);
```

Функция возвращает “файловый” дескриптор, который связан с объектом памяти POSIX. При ошибке возвращает **-1** и устанавливает **errno**

- **name** - имя создаваемого объектра памяти POSIX
- **oflag** - параметры выделения (побитовое ИЛИ)

| Значение | Описание |
|---|---|
| O_RDONLY | открыть только с правами на чтение |
| O_RDWR | открыть с правами на чтение и запись |
| O_CREAT | если объект уже существует, то от флага никакого эффекта. Иначе, объект создается и для него выставляются права доступа в соответствии с mode |
| O_EXCL | установка этого флага в сочетании с O_CREATE приведет к возврату функцией shm_open ошибки, если сегмент общей памяти уже существует |
| O_TRUNC | если объект существует, урезать его размер до нуля |

- **mode** - режим доступа на создаваемый объект памяти

```
#include <unistd.h>
int truncate(const char *path, off_t length);
int ftruncate(int fd, off_t length);
```

Устанавливает размер для файла, задаваемый именем или дескриптором. Возвращает **0** в случае успеха, в случае неудачи **-1** и устанавливает **errno**

```
#include <sys/mman.h>
int shm_unlink(const char *name);
```

## 1.4. Пример работы с разделяемой памятью POSIX

- Пример **pshma.c**

```c
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <fcntl.h>
#include <time.h>

int shmfd=0;
#define PSHM_SIZE 100000000
char const *shm_name="shm0";

int f_error(const char *string) {
        perror(string);
        if (shmfd > 0)
                shm_unlink(shm_name);
        exit(1);
}

int main () {
        int num;
        srand(time(NULL));
        if ((shmfd = shm_open (shm_name, O_CREAT|O_EXCL|O_RDWR, 0600)) == -1 )
                f_error("shm_open()");
        printf("Created POSIX SHMEM with descriptor %d\n",shmfd);
        if (ftruncate(shmfd, PSHM_SIZE ) == -1 )
                f_error("ftruncate()");
        struct stat statbuf;
        if (fstat(shmfd, &statbuf) < 0)
                f_error("fstat()");
        printf("SHMEM file size is %lu\n", statbuf.st_size);
        int *mmf_ptr;
        if ((mmf_ptr = (int *)mmap(0, PSHM_SIZE, PROT_WRITE, MAP_SHARED, shmfd, 0)) == MAP_FAILED)
                f_error("mmap()");
        printf("SHMEM is mmapped to address: %p\n", mmf_ptr);
        for (int i=0; i < 30; i++) {
                num = random() % 1000;
                *mmf_ptr = num;
                printf("The next random number %d\n", num);
                sleep(1);
        }
        munmap(mmf_ptr, PSHM_SIZE);
        close(shmfd);
        shm_unlink(shm_name);
        return 0;
}
```

- Компоновать надо с библиотекой **librt**, чтобы были доступны функции работы с разделяемой

памятью

```
$ cc pshma.c -lrt -o pshma
```

- Получение доступа к объекту разделяемой памяти
  - не использовать O_EXCL при вызове shm_open()

# 2. Сокеты

## 2.1. Общая информация

- **Сокеты** (файловые и сетевые) - с точки зрения ОС - механизм двунаправленной передачи данных

## 2.2. Домены сокетов (семейства адресов)

- **Домены сокетов**, или **семейства адресов** - оперделяет тип описываемого взаимодействия

```
#include <sys/socket.h>
```

| Домен | Описание |
|---|---|
| AF_INET | IPv4 |
| AF_INET6 | IPv6 |
| AF_UNIX | файловые сокеты |
| AF_LOCAL | то же самое, синоним |

- Для каждого домена определен свой формат представления адреса, эти адреса при использовании приводятся к универсальной структуре **sockaddr**

```
#include <sys/socket.h>
struct sockaddr {
    sa_family_t sa_family;  /* семейство адресов */
    char sa_data[14]; /* адрес переменной длины */
};
```

- Для **AF_UNIX(AF_LOCAL)** используется вот такая структура

```
#include <sys/un.h>
struct sockaddr_un {
    sa_family_t sun_family;  /* семейство адресов ==AF_UNIX*/
    char sun_path[108];      /* имя файлового сокета */
};
```

Адрес в данном случае - имя файла

## 2.3. Типы сокетов

- **Тип сокета определяет характеристики взаимодействия**

| Тип | Описание |
|---|---|
| SOCK_DGRAM | Не ориентированы на создание логического соединения, сообщения фиксированной длины, доставка сообщений не гарантируется |
| SOCK_RAW | Интерфейс дейтаграмм к протоколу IP |
| SOCK_STREAM | Ориентированы на создание логического соединения, упорядоченность передачи данных, гарантируется доставка сообщений, двунаправленный поток байтов |

## 2.4. Протоколы взаимодействия

- При определенном сочетании домена и типа подразумевается использование определенного протокола взаимодействия

| Протокол | Сочетание домена и типа |
|---|---|
| IPPROTO_TCP | по умолчанию для AF_INET + SOCK_STREAM |
| IPPROTO_UDP | по умолчанию для AF_INET + SOCK_DGRAM |
| IPPROTO_IP | AF_INET + SOCK_RAW |
| IPPROTO_IPV6 | AF_INET6 + SOCK_RAW |
| IPPROTO_ICMP | AF_INET + SOCK_RAW |
| IPPROTO_RAW | AF_INET + SOCK_RAW |

## 2.5. Дескрипторы сокетов

- С точки зрения приложения, **сокет** - абстракция конечной точки взаимодействия, приложения используют **дискрипторы сокетов** для работы с сокетами, подобно дескрипторам файлов для использования ввода-вывода.
- В **UNIX** дескрипторы сокетов реализованы так же, как дескрипторы файлов, большинство функций низкоуровневого ввода-вывода - **read()**, **write()** работают с дескрипторами сокетов точно так же как и с дескрипторами файлов.
- Создание дескриптора

```
#include <sys/socket.h>
int socket(int domain, int type, int protocol);
```

- **domain** - домен сокета
- **type** - тип сокета
- **protocol** - обычно значение **0**, чтоб выбрать протокол по умолчанию для данного домена и типа

  В случае успеха возвращает дескриптор созданного сокета, в противном случае **0** и устанавливает **errno**

## 2.6. Связывание (bind) сокета

- Для потоковых сокетов - необходимо производить только на серверной стороне соединения. Для датаграмных - на обоих.
- Дескриптор сокета связывается с локальным именем - либо имени файла, либо сетевым адресом, в засисимости от используемого семейства адресов

```
#include <sys/socket.h>
int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

- **sockfd** - дескриптор сокета
- **addr** - указатель на структуру **sockaddr**, структуру с адресом для конкретного семейства адресов необходимо привезти к этому типу
- **addrlen** - размер структуры

```
struct sockaddr_un srv_addr
int saddrlen = sizeof(srv_addr.sun_family) + sizeof(srv_addr.sun_path);
bind(sockfd, (struct sockaddr *) &srv_addr,saddrlen)
```

Пример приведения типов и вызхова **bind()**

## 2.7. Получение сообщения из сокета - recv()

- Функция **recv()**

```
#include <sys/socket.h>
ssize_t recv(int sockfd, void *buf, size_t len, int flags);
```

Возвращает кол-во полученных данных.

- **sockfd** - дескриптор сокета
- **buf** - буффер для размещения полученных данных
- **len** - размер буфера (максимальный размер принятых данных)
- **flags** - флаги

Без установленных флагов **recv()** является практически полным аналогом **read()**

## 2.8. Передача данных в сокет - sendto()

```
# include <sys/socket.h>
ssize_t sendto(int sockfd, const void *buf, size_t len, int flags,
                      const struct sockaddr *dest_addr, socklen_t addrlen);
```

- Передача данных в сокет
- **sockfd** - дескриптор сокета
- **buf** - данные для передачи
- **len** - размер буфера
- **flags** - флаги
- **dest_addr** - сокет-получателя

- **addrlen** - размер структуры с адресом

## 2.9. Пример - сервер и клиент с файловыми сокетами

- Пример **usock-s.c** - сервер

```
#include <sys/socket.h>
#include <sys/un.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>

#define MAXBUFSIZE 256

char buf[MAXBUFSIZE];
char const* socket_file = "./usock";

int pr_exit(char const* str) {
        perror(str);
        exit(errno);
}

int main() {
        struct sockaddr_un srv_addr;
        int sockfd, saddrlen;
        if ((sockfd = socket(AF_UNIX,SOCK_DGRAM,0)) < 0) pr_exit("socket()");
        unlink(socket_file);
        memset(&srv_addr, 0, sizeof(srv_addr));
        srv_addr.sun_family = AF_UNIX;
        strncpy(srv_addr.sun_path, socket_file, sizeof(socket_file));
        saddrlen = sizeof(srv_addr.sun_family) + sizeof(srv_addr.sun_path);
        if (bind(sockfd, (struct sockaddr *) &srv_addr,saddrlen) < 0 ) pr_exit("bind()");
        while (1) {
                if (recv(sockfd, buf, MAXBUFSIZE, 0) < 0 ) pr_exit("recv()");
                write(1,buf, sizeof(buf));
                write(1,"\n",1);
        }
        close(sockfd);
        return 0;
}
```

- Пример **usock-c.c** - клиент

```
#include <sys/socket.h>
#include <sys/un.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>
```

```
char const *msg = "Message from client";
char const *socket_file = "./usock";

int pr_exit(char const* str) {
        perror(str);
        exit(errno);
}

int main() {
        struct sockaddr_un srv_addr, cln_addr;
        int sockfd, saddrlen, caddrlen;
        if ((sockfd = socket(AF_UNIX,SOCK_DGRAM,0)) < 0) pr_exit("socket()");
        memset(&srv_addr, 0, sizeof(srv_addr));
        srv_addr.sun_family = AF_UNIX;
        strncpy(srv_addr.sun_path, socket_file, sizeof(socket_file));
        saddrlen = sizeof(srv_addr.sun_family) + sizeof(srv_addr.sun_path);
        int msglen = strlen(msg);
        if (sendto(sockfd, msg, msglen, 0, (struct sockaddr *)&srv_addr, saddrlen) != msglen ) pr_exit("
        close(sockfd);
        return 0;
}
```

## 2.10. Получение данных от адресуемого абонента - recvfrom()

```
#include <sys/socket.h>
ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,
                        struct sockaddr *src_addr, socklen_t *addrlen);
```

- **src_addr** - по этому указателю записывается адрес источника
- **addrlen** - длина структуры адреса

  Блокирует процесс на время получения данных.

## 2.11. Немного в сторону - генерация временных файлов

- Генерирует уникальное имя на основании шаблона и создает файл. Шаблон должен заканчиваться на XXXXXX (макс 6 штук), эти символы будут заменены на уникальную комбинацию.

```
#include <stdlib.h>
char *mktemp(char *template);
```

Замену производит прямо в параметре (т.е. не должен быть константным) и возвращает указатель на него же.

Использовать не рекомендуется по соображениям безопасности, вместо нее - **mkstemp()** и **mkdtemp()**

```
#include <stdlib.h>
int mkstemp(char *template);
```

Создает уникальное имя из шаблона, создает/открывает файл с таким именем и возвращает его файловый дескриптор.

Имя файла подменяет собой шаблон, поэтому не должно быть константой.

```
#include <stdio.h>
char *tmpnam(char *s);
```

Создает имя для временного файла, сам файл не создается.

- **s** - буфер в котором формируется имя, должен быть длины не меньшей чем **L_tmpnam**

  Имя создатся в каталоге **P_tmpdir**, определен в **stdio.h** как **/tmp**

## 2.12. Пример - двунаправленное взаимодействие с файловыми сокетами

- Пример **usock-s2.c** - сервер

```
#include <sys/socket.h>
#include <sys/un.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>

#define MAXBUFSIZE 256

char buf[MAXBUFSIZE];
char const* socket_file = "./usock";

int pr_exit(char const* str) {
        perror(str);
        exit(errno);
}

int main() {
        struct sockaddr_un srv_addr, cln_addr;
        int sockfd, saddrlen, caddrlen, cpathlen;
        if ((sockfd = socket(AF_UNIX,SOCK_DGRAM,0)) < 0) pr_exit("socket()");
        unlink(socket_file);
        memset(&srv_addr, 0, sizeof(srv_addr));
        srv_addr.sun_family = AF_UNIX;
        strncpy(srv_addr.sun_path, socket_file, sizeof(socket_file));
        saddrlen = sizeof(srv_addr.sun_family) + sizeof(srv_addr.sun_path);
        if (bind(sockfd, (struct sockaddr *) &srv_addr,saddrlen) < 0 ) pr_exit("bind()");
        while (1) {
                caddrlen = sizeof(cln_addr);
                if (recvfrom(sockfd, buf, MAXBUFSIZE, 0, (struct sockaddr *)&cln_addr, &caddrlen) < 0 )
                        pr_exit("recvfrom()");
                if (sendto(sockfd, "OK\n", 3, 0, (struct sockaddr *)&cln_addr, caddrlen) < 0 )
                        pr_exit("sendto()");
                write(1,buf, sizeof(buf));
```

```
                write(1,"[from ",6);
                cpathlen = caddrlen - sizeof(cln_addr.sun_family);
                write(1,cln_addr.sun_path,cpathlen);
                write(1," ]\n",3);
        }
        close(sockfd);
        return 0;
}
```

- Пример **usock-c2.c** - клиент

```
#include <sys/socket.h>
#include <sys/un.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>

#define MAXBUFSIZE 256
char buf[MAXBUFSIZE];
char tmp_file[L_tmpnam];

char const *msg = "Message from client";
char const *socket_file = "./usock";

int pr_exit(char const* str) {
        perror(str);
        exit(errno);
}

int main() {
        struct sockaddr_un srv_addr,cln_addr;
        int sockfd, saddrlen, caddrlen;
        if ((sockfd = socket(AF_UNIX,SOCK_DGRAM,0)) < 0) pr_exit("socket()");
        // адрес сокета сервера
        memset(&srv_addr, 0, sizeof(srv_addr));
        srv_addr.sun_family = AF_UNIX;
        strncpy(srv_addr.sun_path, socket_file, sizeof(socket_file));
        saddrlen = sizeof(srv_addr.sun_family) + sizeof(srv_addr.sun_path);
        // адрес сокета клиента
        memset(&cln_addr, 0, sizeof(cln_addr));
        cln_addr.sun_family = AF_UNIX;
        memset(tmp_file, 0, L_tmpnam);
        if (tmpnam(tmp_file) == NULL ) pr_exit("tmpnam()");
        strncpy(cln_addr.sun_path, tmp_file, sizeof(tmp_file));
        caddrlen = sizeof(cln_addr.sun_family) + sizeof(cln_addr.sun_path);
        if (bind(sockfd, (struct sockaddr *) &cln_addr,caddrlen) < 0 ) pr_exit("bind()");
        int msglen = strlen(msg);
        if (sendto(sockfd, msg, msglen, 0, (struct sockaddr *)&srv_addr, saddrlen) != msglen ) pr_exit("
        if (recv(sockfd, buf, MAXBUFSIZE, 0) < 0 ) pr_exit("recv()");
```

```
        write(1,tmp_file,sizeof(tmp_file));
        write(1, " - ",3);
        write(1,buf,sizeof(buf));
        close(sockfd);
        unlink(tmp_file);
        return 0;
}
```