

Федеральное агентство связи
Сибирский Государственный Университет Телекоммуникаций и
Информатики

Кафедра ТС и ВС

Курсовая работа

По дисциплине: Операционные системы

Выполнил: Зарубин Максим Евгеньевич

Группа: ИА-831

Вариант: 5

Проверила: Моренкова Ольга Ильинична

Новосибирск, 2020 г

Оглавление

Теоретическая часть.....	2
Скриншоты работы программы	9
Код программы.....	10
Список литературы	14

Теоретическая часть

Три типа IPC обычно обобщенно называют System V IPC. Во многих версиях UNIX есть еще одно средство IPC сокет, впервые предложенные в BSD UNIX.1. Сигналы изначально были предложены как средство уведомления об ошибках, но могут использоваться и для элементарного IPC, например, для синхронизации процессов или для передачи простейших команд от одного процесса к другому. Однако использование сигналов в качестве средства IPC ограничено из-за того, что сигналы очень ресурсоемки. Отправка сигнала требует выполнения системного вызова, а его доставка прерывания процесса-получателя и интенсивных операций со стеком процесса для вызова функции обработки и продолжения его нормального выполнения. При этом сигналы слабо информативны и их число весьма ограничено. Поэтому сразу переходим к следующему механизму каналам. Вспомним синтаксис организации программных каналов при работе в командной строке shell:`$cat myfile | sort`

При этом (стандартный) вывод программы `cat`, которая выводит содержимое файла `myfile`, передается на (стандартный) ввод программы `sort`, которая, в свою очередь отсортирует предложенный материал. Таким образом, два процесса обменялись данными. При этом использовался программный канал, обеспечивающий однонаправленную передачу данных между двумя задачами. Хотя в приведенном примере может показаться, что процессы `cat` и `sort` независимы, на самом деле оба этих процесса создаются процессом `shell` и являются родственными. Именованные каналы. Название каналов FIFO происходит от выражения `First In First Out` (первый вошел первый вышел). FIFO очень похожи на каналы, поскольку являются однонаправленным средством передачи данных, причем чтение данных происходит в порядке их записи. Однако в отличие от программных каналов, FIFO имеют имена, которые позволяют независимым процессам получить к этим объектам доступ. Поэтому иногда FIFO также называют именованными каналами.

Как было показано, отсутствие имен у каналов делает их недоступными для независимых процессов. Этот недостаток устранен у FIFO, которые имеют имена. Другие средства межпроцессного взаимодействия, являющиеся более сложными, требуют дополнительных соглашений по именам и идентификаторам. Множество возможных имен объектов конкретного типа межпроцессного взаимодействия называется пространством имен (name space). Имена являются важным компонентом системы межпроцессного взаимодействия (InterProcess Communications IPC) для всех объектов, кроме каналов, поскольку позволяют различным процессам получить доступ к общему объекту. Так, именем FIFO является имя файла именованного канала. Используя условленное имя созданного FIFO два процесса могут обращаться к этому объекту для обмена данными. Для таких объектов IPC, как очереди сообщений, семафоры и разделяемая память, процесс назначения имени является более сложным, чем просто указание имени файла. Имя для этих объектов называется ключом(key) и генерируется функцией `ftok` из двух компонентов: имени файла и идентификатора проекта:

```
#include <sys/types.h>
#include <sys/ipc.h>
key_t ftok(char *filename, char proj);
```

В качестве `filename` можно использовать имя некоторого файла, известное взаимодействующим процессам. Например, это может быть имя программы-сервера. Важно, чтобы этот файл существовал на момент создания ключа. Также нежелательно использовать имя файла, который создается и удаляется в процессе работы распределенного приложения, поскольку при генерации ключа используется номер – `inode` файла. Вновь созданный файл может иметь другой `inode` и впоследствии процесс, желающий иметь доступ к объекту, получит неверный ключ. Пространство имен позволяет создавать и совместно использовать IPC неродственным процессам. Однако для ссылок на уже созданные объекты используются идентификаторы, точно так же, как файловый

дескриптор используется для работы с файлом, открытым по имени. В качестве идентификатора проекта может быть использован любой одиночный символ. Очевидно, что имя файла и идентификатор проекта должны совпадать для всех взаимодействующих процессов. Каждое из перечисленных IPC имеет свой уникальный дескриптор (идентификатор), используемый ОС (ядром) для работы с объектом. Уникальность дескриптора обеспечивается уникальностью дескриптора для каждого из типов объектов (очереди сообщений, семафоры и разделяемая память), т. е. какая-либо очередь сообщений может иметь тот же численный идентификатор, что и разделяемая область памяти (хотя любые две очереди сообщений должны иметь различные идентификаторы). Работа с объектами IPC System V во многом сходна. Для создания или получения доступа к объекту используются соответствующие системные вызовы `getmsgget` для очереди сообщений, `semget` для семафора и `shmget` для разделяемой памяти. Все эти вызовы возвращают дескриптор объекта в случае успеха и -1 в случае неудачи. Отметим, что функции `get` позволяют процессу получить ссылку на объект, которой по существу является возвращаемый дескриптор. Но она не позволяет производить конкретные операции над объектом (помещать или получать сообщения из очереди сообщений, устанавливать семафор или записывать данные в разделяемую память. Все функции `get` в качестве аргументов используют ключ `key` и флажки создания объекта `ipcflag`. Остальные аргументы зависят от конкретного типа объекта. Переменная `ipcflag` определяет права доступа к объекту `PERM`, а также указывает, создается ли новый объект или требуется доступ к существующему. Последнее определяется комбинацией (или отсутствием) флажков `IPC_CREAT` и `IPC_EXCL`. Права доступа к объекту указываются набором флажков доступа, подобно тому, как это делается для файлов. Работа с объектами IPC System V во многом похожа на работу с файлами UNIX. Одним из различий является то, что файловые дескрипторы имеют значимость в контексте процесса, в то время как значимость дескрипторов объектов IPC распространяется на всю систему. Так файловый дескриптор 3 одного процесса в общем случае никак не связан с

дескриптором 3 другого неродственного процесса (т. е. эти дескрипторы ссылаются на различные файлы). Иначе обстоит дело с дескрипторами объектов IPC. Все процессы, использующие, скажем, одну очередь сообщений, получают одинаковые дескрипторы этого объекта. Для каждого из объектов IPC ядро поддерживает соответствующую структуру данных, отличную для каждого типа объекта IPC (очереди сообщений, семафора или разделяемой памяти). Общей у этих данных является структура `ipc_perm`, описывающая права доступа к объекту, подобно тому, как это делается для файлов. Права доступа (как и для файлов) определяют возможные операции, которые может выполнять над объектом конкретный процесс (получение доступа к существующему объекту, чтение, запись и удаление). Заметим, что система не удаляет созданные объекты IPC даже тогда, когда ни один процесс не пользуется ими. Удаление созданных объектов является обязанностью процессов, которым для этого предоставляются соответствующие функции управления `msgctl`, `semctl`, `shmctl`. С помощью этих функций процесс может получить и установить ряд полей внутренних структур, поддерживаемых системой для объектов IPC, а также удалить созданные объекты. Безусловно, как и во многих других случаях использования объектов IPC процессы предварительно должны "договориться", какой процесс и когда удалит объект. Чаще всего, таким процессом является сервер.

Как уже обсуждалось, очереди сообщений являются составной частью UNIX System V. Они обслуживаются операционной системой, размещаются в адресном пространстве ядра и являются разделяемым системным ресурсом. Каждая очередь сообщений имеет свой уникальный идентификатор. Процессы могут записывать и считывать сообщения из различных очередей. Процесс, пославший сообщение в очередь, может не ожидать чтения этого сообщения каким-либо другим процессом. Он может закончить свое выполнение, оставив в очереди сообщение, которое будет прочитано другим процессом позже. Данная возможность позволяет процессам обмениваться структурированными

данными, имеющими следующие атрибуты:- Тип сообщения (позволяет мультиплексировать сообщения в одной очереди)- Длина данных сообщения в байтах (может быть нулевой)

- Собственно данные (если длина ненулевая, могут быть структурированными) Очередь сообщений хранится в виде внутреннего однонаправленного связанного списка в адресном пространстве ядра. Для каждой очереди ядро создает заголовок очереди (`msqid_ds`), где содержится информация о правах доступа к очереди (`msg_perm`), ее текущем состоянии (`msg_cbytes` число байтов и `msg_qnum` число сообщений в очереди), а также указатели на первое (`msg_first`) и последнее (`msg_last`) сообщения, хранящиеся в виде связанного списка. Каждый элемент этого списка является отдельным сообщением. Для создания новой очереди сообщений или для доступа к существующей используется системный вызов `msgget`:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgget (key_t key, int msgflag);
```

Функция возвращает дескриптор объекта-очереди, либо -1 в случае ошибки. Подобно файловому дескриптору, этот идентификатор используется процессом для работы с очередью сообщений. В частности, процесс может:

Помещать в очередь сообщения с помощью функции `msgsnd`;

Получать сообщения определенного типа из очереди с помощью функции `msgrcv`;

Управлять сообщениями с помощью функции `msgctl`.

Перечисленные системные вызовы манипулирования сообщениями имеют следующий вид:

```
#include <sys/ types.h>
#include <sys/ ipc. h>
```

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

int msgsnd (int msqid, const void *msgp, size_t msgsz, int msgflg);

int msgrcv (int msqid, void *msgp, size_t msgsz, long msgtyp, int msgflg);

int msgctl (int msqid, int cmd, struct msgid_ds * buf);
```

Здесь `msqid` является дескриптором объекта, полученного в результате вызова `msgget`. Параметр `msgp` указывает на буфер, содержащий тип сообщения и его данные, размер которого равен `msgsz` байт. Буфер имеет следующие поля: `long msgtype; /*тип сообщения*/char msgtext []; /*данные сообщения*/`

Аргумент `msgtyp` указывает на тип сообщения и используется для их выборочного получения. Если `msgtyp` равен 0, функция `msgrcv` получит первое сообщение из очереди. Если величина `msgtyp` выше 0, будет получено первое сообщение указанного типа. Если `msgtyp` меньше 0, функция `msgrcv` получит сообщение с минимальным значением типа, меньше или равного абсолютному значению `msgtyp`. Очереди сообщений обладают весьма полезным свойством - в одной очереди можно мультиплексировать сообщения от различных процессов. Для демультимплексирования используется атрибут `msgtype`, на основании которого любой процесс может фильтровать сообщения с помощью функции `msgrcv`, как это было показано выше. Функция `msgctl` выполняет контрольную операцию, заданную в `cmd`, над очередью сообщений `msqid`.

Скопировать информацию из структуры данных очереди сообщений в структуру с адресом `buf` (причем, у пользователя должны быть права на чтение очереди сообщений). `IPC_SET` Записать значения некоторых элементов структуры `msqid_ds`, адрес которой указан в `buf`, в структуру данных из очереди сообщений, обновляя при этом его поле `msg_ctime`. Предполагаемые элементы заданной пользователем структуры `struct msqid_ds`, адрес которой указан в `buf`, являются следующими:

`msg_perm.uid`, `msg_perm.gid`, `msg_perm.mode` /* только неосновные 9 битов */,
`msg_qbytes`

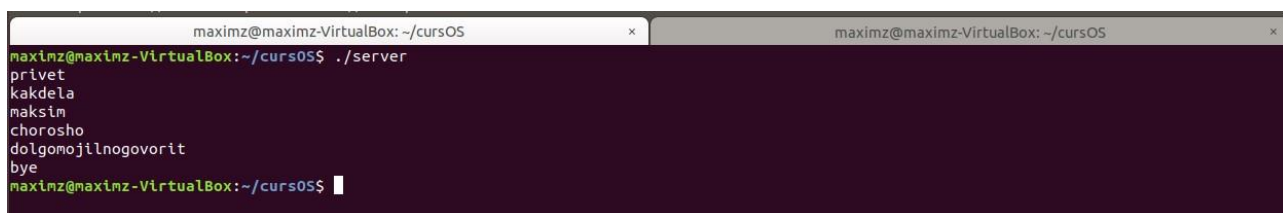
Эффективный идентификатор пользователя процесса, вызывающего эту функцию, должен принадлежать либо root, либо создателю или владельцу очереди сообщений. Только суперпользователь может устанавливать значение `msg_qbytes` большим, чем `MSGMNB`. Немедленно удалить очередь сообщений и структуры его данных, "разбудив" все процессы, ожидающие записи или чтения этой очереди (при этом функция возвращает ошибку, а переменная `errno` приобретает значение `EIDRM`). Эффективный идентификатор пользователя процесса, вызывающего эту функцию, должен принадлежать либо root, либо создателю или владельцу очереди сообщений. Рассмотрим типичную ситуацию взаимодействия процессов, когда серверный процесс обменивается данными с несколькими клиентами. Свойство мультиплексирования позволяет использовать для такого обмена одну очередь сообщений. Для этого сообщениям, направляемым от любого из клиентов серверу, будем присваивать значение типа, скажем, равным 1. Если в теле сообщения клиент каким-либо образом идентифицирует себя (например, передает свой PID), то сервер сможет передать сообщение конкретному клиенту, присваивая тип сообщения равным этому идентификатору. Поскольку функция `msgrcv` позволяет принимать сообщения определенного типа (типов), сервер будет принимать сообщения с типом 1, а клиенты - сообщения с типами, равными идентификаторам их процессов.

Скриншоты работы программы



```
maximz@maximz-VirtualBox: ~/cursOS
maximz@maximz-VirtualBox:~/cursOS$ ./client
Введите строку: privet
tevirp
Введите строку: kakdela
aledkak
Введите строку: maksim
miskam
Введите строку: chorosho
ohsorohc
Введите строку: dolgomojlnogovorit
tirovogonlijomoglod
Введите строку: bye
bye
maximz@maximz-VirtualBox:~/cursOS$
```

Рис.1. Работа клиента.



```
maximz@maximz-VirtualBox: ~/cursOS
maximz@maximz-VirtualBox:~/cursOS$ ./server
privet
kakdela
maksim
chorosho
dolgomojlnogovorit
bye
maximz@maximz-VirtualBox:~/cursOS$
```

Рис.2. Работа сервера.

Код программы

Mesg.h:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#define MAXBUFF 80
#define PERM 0666

typedef struct our_msgbuf
{
    long mtype;
    char buff[MAXBUFF];
}
Message;
```

Client.c:

```
#include "../mesg.h"

main()
{
    Message message;
    key_t key;
    int msgid, length, k;
    message.mtype = 1L;
    char mass[80];
    if ( (key = ftok("server", 'A')) < 0)
    {
        printf("Невозможно получить ключ\n");
        exit(1);
    }

    if ((msgid = msgget(key, 0)) < 0)
    {
        printf("Невозможно получить доступ к очереди\n");
        exit(1);
    }

    while(1)
    {
        printf("Введите строку: ");
        scanf("%s", mass);
        if (strcmp(mass, "bye") == 0)
        {
            if ((length = sprintf(message.buff, "bye")) < 0)
```

```

    {
        printf("Ошибка копирования в буфер\n");
        exit(1);
    }

    if (msgsnd(msgid, (void *) &message, length, 0) != 0)
    {
        printf("Ошибка записи сообщения в очередь\n");
        exit(1);
    }
}
else
{
    if ((length = sprintf(message.buff, mass)) < 0)
    {
        printf("Ошибка копирования в буфер\n");
        exit(1);
    }

    if (msgsnd(msgid, (void *) &message, length, 0) != 0)
    {
        printf("Ошибка записи сообщения в очередь\n");
        exit(1);
    }
}
k=msgrcv(msgid, &message, sizeof(message), message.mtype, 0);
if (k > 0)
{
    if (write(1, message.buff, k) != k)
    {
        printf("Ошибка вывода\n");
        exit(1);
    }
}
else
{
    printf("Ошибка чтения сообщения\n");
    exit(1);
}
printf("\n");
if(strcmp(mass, "bye") == 0)
{
    break;
}
}
if (msgctl(msgid, IPC_RMID, 0) < 0)
{
    printf("Ошибка удаления очереди\n");
    exit(1);
}
exit(0);
}

```

Server.c:

```
#include "../mesg.h"

void invert(char *str)
{
    char c;
    int i, len, f;
    len=strlen(str);
    f=len/2;
    for (i=0; i<f; i++)
    {
        c=str[i];
        str[i]=str[len-1-i];
        str[len-1-i]=c;
    }
}

main ()
{
    Message message;
    key_t key;
    int msgid, length, n;
    char mass[80], c;

    if ((key = ftok("server", 'A')) < 0)
    {
        printf("Невозможно получить ключ\n");
        exit(1);
    }

    message.mtype=1L;

    if ((msgid = msgget(key, PERM | IPC_CREAT)) < 0)
    {
        printf("Невозможно создать очередь\n");
        exit(1);
    }

    while(1)
    {
        n=msgrcv(msgid, &message, sizeof(message), message.mtype,
0);
        if (n > 0)
        {
            if (write(1, message.buf, n) != n)
            {
                printf("Ошибка вывода\n");
                exit(1);
            }
        }
        else
    }
```

```

    {
        printf("Ошибка чтения сообщения\n");
        exit(1);
    }

    printf("\n");

    if (strcmp(message.buff, "bye") == 0)
    {
        if ((length = sprintf(message.buff, "bye")) < 0)
        {
            printf("Ошибка копирования в буфер\n");
            exit(1);
        }

        if (msgsnd(msgid, (void *) &message, length, 0) != 0)
        {
            printf("Ошибка записи сообщения в очередь\n");
            exit(1);
        }
    }
    else
    {
        invert(message.buff);
        if ((length = sprintf(mass, message.buff)) < 0)
        {
            printf("Ошибка копирования в буфер\n");
            exit(1);
        }

        if (msgsnd(msgid, (void *) &message, length, 0) != 0)
        {
            printf("Ошибка записи сообщения в очередь\n");
            exit(1);
        }
    }
    if (strcmp(message.buff, "bye") == 0)
    {
        exit(0);
    }
    memset(message.buff, 0, 80);
}

```

Список литературы

1. Linux: Полное руководство [Электронный ресурс]. – Режим доступа: [<https://it.wikireading.ru/14109>], свободный – (15.12.2020).
2. Лекция 6: Очереди сообщений в UNIX [Электронный ресурс]. – Режим доступа: [<https://intuit.ru/studies/courses/2249/52/lecture/1560>], свободный – (15.12.2020).
3. Очереди сообщений [Электронный ресурс]. – Режим доступа: [<http://masters.donntu.org/2005/fvti/lukyanov/library/ipc/msgq.html>], свободный – (15.12.2020).