

МТИ. ПРИЕМНАЯ КОМИССИЯ. ИДЕТ НАБОР.

БОЛЬШЕ НА [TL.MTI.EDU.RU](https://tl.mti.edu.ru)

Учитесь и получайте официальные документы БЕСПЛАТНО. Вы можете поддержать наш проект



[Регистрация](#) [Вход](#)

[Учеба](#) [Академии](#) [Учителя](#) [Рейтинг](#) [Вопросы](#) [Магазин](#)

[Сведения об образовательной организации](#)

[Курсы](#) | [Школа](#) | [Мини-МБА](#) | [Профессиональная переподготовка](#) | [Повышение квалификации](#) | [Сертификации](#)

0 курсе

[Информация](#)

[Глоссарий](#)

[Дипломы](#)

[Вопросы и ответы](#)

[Студенты](#)

[Рейтинг выпускников](#)

[Мнения](#)

[Литература](#)

[Учебные программы](#)

[План занятий](#)

[Экзамен экстерном](#)

[Лекция 1](#)

[Тест 1](#)

[Лекция 2](#)

[Тест 2](#)

[Лекция 3](#)

[Тест 3](#)

[Лекция 4](#)

[Тест 4](#)

[Лекция 5](#)

[Тест 5](#)

[Лекция 6](#)

[Тест 6](#)

[Лекция 7](#)

[Лекция 8](#)

[Тест 7](#)

Академия Intel: Основы операционных систем. Практикум

[+]

Реклама



[Записаться](#) | [Вам нравится?](#) Нравится 66 студентам | [Поделиться](#) | [Поддержать курс](#) | [Скачать электронную книгу](#)

Лекция 3: Организация взаимодействия процессов через `pipe` и `FIFO` в UNIX

[A](#) | [версия для печати](#)

[< Лекция 2](#) || [Лекция 3: 1 2 3 4 5 6](#) || [Лекция 4 >](#)

Понятие FIFO. Использование системного вызова `mknod()` для создания FIFO. Функция `mkfifo()`

Как мы выяснили, доступ к информации о расположении `pipe`'а в операционной системе и его состоянии может быть осуществлен только через *таблицу открытых файлов процесса*, создавшего `pipe`, и через унаследованные от него *таблицы открытых файлов* процессов-потомков. Поэтому изложенный выше механизм обмена информацией через `pipe` справедлив лишь для родственных процессов, имеющих общего прародителя. *Создавшего системный вызов `pipe()`*, или для таких процессов и самого прародителя и не может использоваться для потокового общения с другими процессами. В операционной системе UNIX существует возможность использования `pipe`'а для взаимодействия других процессов, но ее реализация достаточно сложна и лежит далеко за пределами наших занятий.

Для организации потокового взаимодействия любых процессов в операционной системе UNIX применяется средство связи, получившее название **FIFO** (от First Input First Output) или именованный `pipe`. FIFO во всем подобен `pipe`'у, за одним исключением: данные о

Реклама

РЕКЛАМА

Яндекс Директ



5000 руб.

Получите промокод 5000 рублей на первую рекламную кампанию в Яндекс.Директе

Получить

[ответить](#)

РЕКЛАМА

Яндекс Практикум

Учиться новому во взрослом возрасте — сложно



Правда или миф? Расскажет выпускник Практикума



Лекция 9
Тест 8
Лекция 10
Тест 9
Экзамен

Спонсор: Intel



Вы можете [поддержать](#)
этот курс.

расположении *FIFO* в адресном пространстве ядра и его состоянии процессы могут получать не через родственные связи, а через файловую систему. Для этого при создании именованного *pip*'а на диске заводится *файл* специального типа, обращаясь к которому процессы могут получить интересующую их информацию. Для создания *FIFO* используется системный вызов ***mknod()*** или существующая в некоторых версиях UNIX функция ***mkfifo()***.

Следует отметить, что при их работе не происходит действительного выделения области адресного пространства операционной системы под именованный *pipe*, а только заводится файл-метка, существование которой позволяет осуществить реальную организацию *FIFO* в памяти при его открытии с помощью уже известного нам системного вызова ***open()***.

После открытия именованный *pipe* ведет себя точно так же, как и неименованный. Для дальнейшей работы с ним применяются системные вызовы ***read()***, ***write()*** и ***close()***. Время существования *FIFO* в адресном пространстве ядра операционной системы, как и в случае с *pip*'ом, не может превышать время жизни последнего из использовавших его процессов. Когда все процессы, работающие с *FIFO*, закрывают все *файловые дескрипторы*, ассоциированные с ним, система освобождает ресурсы, выделенные под *FIFO*. Вся непрочитанная *информация* теряется. В то же время *файл-метка* остается на диске и может использоваться для новой реальной организации *FIFO* в дальнейшем.

Использование системного вызова **mknod** для создания FIFO

Прототип системного вызова

```
#include <sys/stat.h>
#include <unistd.h>
int mknod(char *path, int mode, int dev);
```

Описание системного вызова

Нашей целью является не полное описание системного вызова ***mknod***, а только описание его использования для создания FIFO. Поэтому мы будем рассматривать не все возможные варианты задания параметров, а только те из них, которые соответствуют этой специфической деятельности.

Параметр ***dev*** является несущественным в нашей ситуации, и мы будем всегда задавать его равным **0**.

Параметр ***path*** является указателем на строку, содержащую полное или *относительное имя файла*, который будет являться меткой FIFO на диске. Для успешного создания FIFO файла с таким именем перед вызовом существовать не должно.

Параметр ***mode*** устанавливает атрибуты прав доступа различных категорий пользователей к FIFO. Этот параметр задается как результат побитовой операции "или" значения ***S_IFIFO***, указывающего, что системный вызов должен создать FIFO, и некоторой суммы следующих восьмеричных значений:

- 0400** – разрешено чтение для пользователя, создавшего FIFO;
- 0200** – разрешена запись для пользователя, создавшего FIFO;
- 0040** – разрешено чтение для группы пользователя, создавшего FIFO;
- 0020** – разрешена запись для группы пользователя, создавшего FIFO;
- 0004** – разрешено чтение для всех остальных пользователей;
- 0002** – разрешена запись для всех остальных пользователей.

При создании FIFO реально устанавливаемые права доступа получаются из стандартной комбинации параметра ***mode*** и *маски создания файлов текущего процесса* ***umask***, а именно – они равны ***(0777 & mode) & ~umask***.

Возвращаемые значения

При успешном создании FIFO системный вызов возвращает значение **0**, при неуспешном – отрицательное значение.

Функция **mkfifo**

Прототип функции

Реклама

РЕКЛАМА

Яндекс Директ

✂
5000 руб.

Получите промокод 5000
рублей на первую рекламную
кампанию в Яндекс.Директе

Получить

РЕКЛАМА • ОТМ

```
#include <sys/stat.h>
#include <unistd.h>
int mkfifo(char *path, int mode);
```

Описание функции

Функция `mkfifo` предназначена для создания FIFO в операционной системе.

Параметр `path` является указателем на строку, содержащую полное или *относительное имя файла*, который будет являться меткой FIFO на диске. Для успешного создания FIFO файла с таким именем перед вызовом функции не должно существовать.

Параметр `mode` устанавливает атрибуты прав доступа различных категорий пользователей к FIFO. Этот параметр задается как некоторая сумма следующих восьмеричных значений:

- 0400 – разрешено чтение для пользователя, создавшего FIFO;
- 0200 – разрешена запись для пользователя, создавшего FIFO;
- 0040 – разрешено чтение для группы пользователя, создавшего FIFO;
- 0020 – разрешена запись для группы пользователя, создавшего FIFO;
- 0004 – разрешено чтение для всех остальных пользователей;
- 0002 – разрешена запись для всех остальных пользователей.

При создании FIFO реально устанавливаемые права доступа получаются из стандартной комбинации параметра `mode` и *маски создания файлов текущего процесса* `umask`, а именно – они равны `(0777 & mode) & ~umask`.

Возвращаемые значения

При успешном создании FIFO функция возвращает значение `0`, при неуспешном – отрицательное значение.

Важно понимать, что *файл типа FIFO* не служит для размещения на диске информации, которая записывается в именованный *pipe*. Эта информация располагается внутри адресного пространства операционной системы, а *файл* является только меткой, создающей предпосылки для ее размещения.

Не пытайтесь просмотреть содержимое этого файла с помощью Midnight Commander (mc)!!! Это приведет к его глубокому зависанию!

Особенности поведения вызова `open()` при открытии FIFO

Системные вызовы `read()` и `write()` при работе с *FIFO* имеют те же особенности поведения, что и при работе с *pip*'ом. Системный вызов `open()` при открытии *FIFO* также ведет себя несколько иначе, чем при открытии других типов файлов, что связано с возможностью блокирования выполняющих его процессов. Если *FIFO* открывается только для чтения, и флаг `O_NDELAY` не задан, то процесс, осуществивший *системный вызов*, блокируется до тех пор, пока какой-либо другой процесс не откроет *FIFO* на *запись*. Если флаг `O_NDELAY` задан, то возвращается *значение файлового дескриптора*, ассоциированного с *FIFO*. Если *FIFO* открывается только для записи, и флаг `O_NDELAY` не задан, то процесс, осуществивший *системный вызов*, блокируется до тех пор, пока какой-либо другой процесс не откроет *FIFO* на чтение. Если флаг `O_NDELAY` задан, то констатируется возникновение ошибки и возвращается *значение -1*. Задание флага `O_NDELAY` в параметрах системного вызова `open()` приводит к тому, что процессу, открывшему *FIFO*, запрещается *блокировка* при выполнении последующих операций чтения из этого потока данных и записи в него.

Прогон программы с FIFO в родственных процессах

Для иллюстрации взаимодействия процессов через *FIFO* рассмотрим такую программу:

```
/* Программа 05-4.с, осуществляющая однонаправленную связь через
FIFO между процессом-родителем и процессом-ребенком */
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
```

Реклама

РЕКЛАМА

Яндекс Директ

✂

5000 руб.

Получите промокод 5000
рублей на первую рекламную
кампанию в Яндекс.Директе

Получить

РЕКЛАМА

NINJA
PIZZA

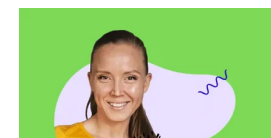
Акция дня!

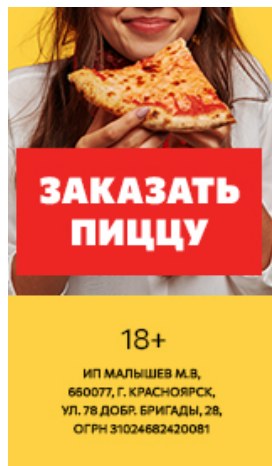
1+1=4

Закажи 2 пиццы
и получи 2 пиццы
в подарок!

Реклама

РЕКЛАМА





```
#include <stdio.h>
int main(){
    int fd, result;
    size_t size;
    char resstring[14];
    char name[]="aaa.fifo";
    /* Обнуляем маску создания файлов текущего процесса для того,
    чтобы права доступа у создаваемого FIFO точно соответствовали
    параметру вызова mknod() */
    (void)umask(0);
    /* Попытаемся создать FIFO с именем aaa.fifo в текущей
    директории */
    if(mknod(name, S_IFIFO | 0666, 0) < 0){
        /* Если создать FIFO не удалось, печатаем об этом
        сообщение и прекращаем работу */
        printf("Can't create FIFO\n");
        exit(-1);
    }
    /* Порождаем новый процесс */
    if((result = fork()) < 0){
        /* Если создать процесс не удалось, сообщаем об этом и
        завершаем работу */
        printf("Can't fork child\n");
        exit(-1);
    } else if (result > 0) {
        /* Мы находимся в родительском процессе, который будет
        передавать информацию процессу-ребенку. В этом процессе
        открываем FIFO на запись.*/
        if((fd = open(name, O_WRONLY)) < 0){
            /* Если открыть FIFO не удалось, печатаем об этом
            сообщение и прекращаем работу */
            printf("Can't open FIFO for writing\n");
            exit(-1);
        }
        /* Пробуем записать в FIFO 14 байт, т.е. всю строку
        "Hello, world!" вместе с признаком конца строки */
        size = write(fd, "Hello, world!", 14);
        if(size != 14){
            /* РЕКЛАМА • ОТМ
            Если записалось меньшее количество байт, то сообщаем
            об ошибке и завершаем работу */
            printf("Can't write all string to FIFO\n");
            exit(-1);
        }
    }
}
```



РЕКЛАМА

online.swiftbook.ru

Стать программистом на Swift с нуля – это реально!

[Узнать больше](#)

рублей на первую рекламную кампанию в Яндекс.Директе

[Получить](#)

```

}
/* Закрываем входной поток данных и на этом родитель
прекращает работу */
close(fd);
printf("Parent exit\n");
} else {
/* Мы находимся в порожденном процессе, который будет
получать информацию от процесса-родителя. Открываем
FIFO на чтение.*/
if((fd = open(name, O_RDONLY)) < 0){
/* Если открыть FIFO не удалось, печатаем об этом
сообщение и прекращаем работу */
printf("Can't open FIFO for reading\n");
exit(-1);
}
/* Пробуем прочитать из FIFO 14 байт в массив, т.е.
всю записанную строку */
size = read(fd, resstring, 14);
if(size < 0){
/* Если прочитать не смогли, сообщаем об ошибке
и завершаем работу */
printf("Can't read string\n");
exit(-1);
}
/* Печатаем прочитанную строку */
printf("%s\n",resstring);
/* Закрываем входной поток и завершаем работу */
close(fd);
}
return 0;
}

```

Листинг 5.4. Программа 05-4.c, осуществляющая однонаправленную связь через FIFO между процессом-родителем и процессом-ребенком

Наберите программу, откомпилируйте ее и запустите на *исполнение*. В этой программе информацией между собой обмениваются процесс-родитель и процесс-ребенок. Обратите внимание, что повторный *запуск* этой программы приведет к ошибке при попытке создания *FIFO*, так как *файл* с заданным именем уже существует. Здесь нужно либо удалять его перед каждым прогоном программы с диска вручную, либо после первого запуска модифицировать исходный текст, исключив из него все, связанное с системным вызовом *mknod()*. С системным вызовом, предназначенным для удаления файла при работе процесса, мы познакомимся позже (на семинарах 11-12) при изучении файловых систем.

Написание, компиляция и запуск программы с FIFO в неродственных процессах

Для закрепления полученных знаний напишите на базе предыдущего примера две программы, одна из которых пишет информацию в *FIFO*, а вторая – читает из него, так чтобы между ними не было ярко выраженных родственных связей (т.е. чтобы ни одна из них не была потомком другой).

Реклама

РЕКЛАМА

Яндекс Директ

✂

5000 руб.

Получите промокод 5000
рублей на первую рекламную
кампанию в Яндекс.Директе

Получить

РЕКЛАМА

III долями

0+

Узнайте
больше
о Долях

Подробнее

РЕКЛАМА • ОТМ

Неработающий пример для связи процессов на различных компьютерах

Если у вас есть возможность, найдите два компьютера, имеющих разделяемую файловую систему (например, смонтированную с помощью *NFS*), и запустите на них программы из предыдущего раздела так, чтобы каждая *программа* работала на своем компьютере, а *FIFO* создавалось на разделяемой файловой системе. Хотя оба процесса видят один и тот же *файл с типом FIFO*, взаимодействия между ними не происходит, так как они функционируют в физически разных адресных пространствах и пытаются открыть *FIFO* внутри различных операционных систем.

[Дальше >>](#)[< Лекция 2](#) || [Лекция 3: 1 2 3 4 5 6](#) || [Лекция 4 >](#)

АО «Тинькофф Банк»,
лицензия №2673

РЕКЛАМА



Получите \$52

Товары на Таобао от \$1



Реклама

Реклама


Крутой цикл на АТ! Законченная история!

Пять книг, иллюстрации, восторженные отзывы!

author.today

[Узнать больше](#)

РЕКЛАМА



ДЕПАРТАМЕНТ
ПРЕДПРИНИМАТЕЛЬСТВА
И ИННОВАЦИОННОГО РАЗВИТИЯ
ГОРОДА МОСКВЫ

АГЕНТСТВО
ИННОВАЦИЙ
МОСКВЫ

ОНЛАЙН-ХАКАТОН

Создай новые решения для города

Реклама

РЕКЛАМА

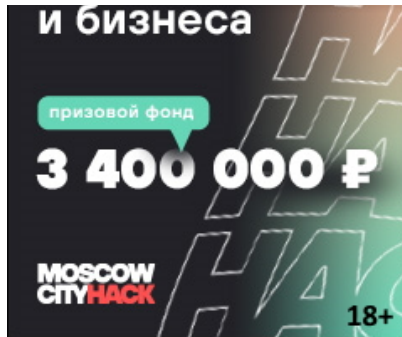
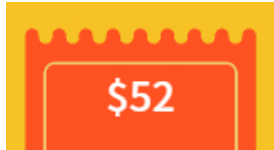
Яндекс Директ

✂

5000 руб.

Получите промокод 5000
рублей на первую рекламную
кампанию в Яндекс.Директе

[Получить](#)



© НОУ «ИНТУИТ»,
2003 – 2022

[Пользовательское соглашение](#) | [Политика конфиденциальности](#) | [Реклама на сайте](#) | [Напишите нам](#)

Телефон: +7 (499) 253-9312, WhatsApp, Telegram, Viber: +7 (977) 954-84-50, факс: +7 (499) 253-9310, e-mail: info@intuit.ru, Skype: Intu:

Реклама

РЕКЛАМА

Яндекс Директ

✂
5000 руб.

Получите промокод 5000
рублей на первую рекламную
кампанию в Яндекс.Директе

Получить

РЕКЛАМА • ОТМ