Пензенский государственный университет

Кафедра «МОиПЭВМ»

Методические указания к лабораторным работам   
по дисциплине «**Операционные системы и сети**»

**Пенза 2014**

Работы выполняются в среде операционной системы Linux. для выполнения работ необходимо использовать приведенную в Библиографическом списке литературу, конспект курса лекций, а также справочные материалы из документации на систему, полученные с помощью системной утилиты man.

Вся доступная при использовании утилиты man информация имеет многоуровневую структуру. Если документ имеет несколько таких уровней, то это отмечается в конце описания документа. Для того чтобы получить соответствующую информацию по команде операционной системы на первом уровне необходимо набрать строку man [<уровень>] <имя команды>. Номер первого уровня задается по умолчанию.

При вызове утилиты информация выдается на экран постранично. Для перемещений по тексту можно использовать стрелки вниз и вверх. Листание по страницам вперед выполняется по клавишам f или пробел. Листание назад по клавише b. Выход их просмотра документа выполняется по клавише q.

Утилита позволяет организовать поиск внутри текста руководства. Для того чтобы найти слово с просмотром текста вперед необходимо ввести /<слово>, для поиска назад ‑ ?<слово>. Продолжение поиска по заданному образцу производится по клавише n.

Для получения протоколов результатов выполнения работ следует перенаправлять данные из стандартного потока вывода в файл.

Требования к оформлению работ

По каждой лабораторной работе составляется отчет, который должен содержать:

* титульный лист;
* название и цель работы;
* лабораторное задание;
* для программы ‑ описание данных и при необходимости описание структуры программы;
* текст программы или скрипта;
* результаты выполнения команд, скриптов и программ;
* выводы по результатам выполнения работы.

Отчет может представляться в виде твердой копии или в виде файла в формате редакторов LibreOffice или MS Word.

Лабораторная работа № 1. Вход в систему и регистрация пользователей

**Цель работы**: познакомиться с особенностями загрузки операционной системы Linux, входа в систему и регистрации пользователя.

**Методические указания**

В операционной системе Linux файлы организованы в виде древовидной структуры, называемой файловой системой. Корневой каталог дерева имеет имя "**/**". Имена всех остальных файлов содержат путь по дереву, начинающийся с имени главного каталога (абсолютное или полное путевое имя файла). При обращении к файлам текущего каталога можно использовать относительные имена, которые содержат имена файлов в самом текущем каталоге и путевые имена в каталогах, расположенных по дереву ниже текущего. В системе используются специальные имена для обозначения текущего каталога ‑ "**.**" и родительского каталога ‑ "**..**". Для упрощения работы можно использовать специальный символ "**~**". Этот символ обозначает абсолютное путевое имя начального каталога.

Особенностью операционной системы Linux является то, что все файлы в каталогах располагаются в соответствии со своими типами. Рассмотрим примеры размещения файлов в каталогах.

**/bin** ‑ в этом каталоге размещаются файлы, реализующие команды и утилиты системы общего пользования.

**/dev** ‑ этот каталог содержит специальные файлы являющиеся интерфейсом для доступа к периферийным устройствам.

**/etc** ‑ здесь размещаются системные конфигурационные файлы и утилиты администрирования.

**/lib** ‑ данный каталог служит для хранения библиотечных файлов систем программирования.

**/mnt** ‑ стандартный каталог для временного связывания (монтирования) других файловых систем для получения единого дерева.

**/home** ‑ каталог для размещения каталогов пользователей.

**/usr** ‑ здесь размещаются подкаталоги содержащие файлы различных приложений, их назначение также определяется типами файлов.

**Порядок выполнения работы**

После загрузки системы запустить терминал.

Просмотреть страницу руководства для команде ls, выполнив команду вызова справочной системы man ls.

Выполнить команду ls, которая позволяет просмотреть содержимое текущего каталога.

Выполнить команду просмотра текущей даты и времени date.

Запустить программу просмотра файловой системы.

Проанализировать содержимое текущего каталога и место каталога пользователя в смонтированной файловой системе.

Проанализировать характер размещения файлов различного назначения в каталогах.

Завершить сеанс работы.

Лабораторная работа № 2. Команды Shell

**Цель работы**: изучить особенности формирования и выполнения некоторых команд оболочки Bourn shell операционной системы Linux.

**Методические указания**

Интерфейс операционной системы представлен командным интерпретатором shell. В лабораторном практикуме будет использоваться командный интерпретатор Bourn again shell (bash). Выполнение команд должно производиться в терминале. Для работы с текстовыми файлами можно использовать любой текстовый редактор, например gedit или kate.

В общем случае команда имеет следующий вид:

**<команда> <ключи> <аргументы>**

Ключ определяет режим выполнения команды и представляет собой символ, перед которым указывается знак **-**, например, -d. Возможно объединение нескольких ключей в группу: -al.

Команда просмотра содержимого каталога ls выдает список имен файлов и каталогов. Общий вид команды

**ls [<ключи>] [<аргументы>]**

Команда cat выводит содержимое файла на экран терминала. Команда more выводит содержимое файла поэкранно. Для управления просмотром так же как при работе с утилитой man можно использовать клавиши f, b и q.

Команда pwd выдает на экран абсолютное путевое имя текущего каталога. Команда cd изменяет текущий каталог. При использовании команды без параметра выполняется переход в домашний каталог пользователя.

Команда копирования cp файлов имеет формат:

**cp [ключи] <исходный файл> <результирующий файл>**

Команда перемещения (переименования) имеет формат:

**mv [<ключи>] <текущее имя файла> <новое имя файла>**,

для перемещения файла в другой каталог используется команда:

**mv [<ключи>] <имя файла> <имя каталога>**

В файловой системе ОС Linux один файл может иметь несколько имен. Этот механизм основан на том, что с помощью специальной команды ln могут создаваться специальные структуры данных ‑ связи. Связи бывают жесткими и символическими. В первом случае связь хранится в управляющих метаданных файла. Во втором ‑ в каталоге создается специальный файл, содержащий имя целевого файла.

Команда создания связи ln. Формат команды:

**ln [<ключи>] <имя файла> <имя файла>**

Команда ln с ключом -s создает символическую связь.

Команда удаления файла rm

**rm [<ключи>] <имя файла>**

Команда удаляет имя файла из каталога, при этом уменьшается на 1 счетчик жестких ссылок файла. Если счетчик становиться равным 0, то файл уничтожается.

При использовании ключа -i запрашивается подтверждение на выполнение операции. Команда rmможет использоваться и для удаления каталогов, для этого она должна использоваться с ключом -r. При этом сначала удаляется все содержимое каталога, а затем и сам каталог. Для удаления пустых каталогов служит команда rmdir.

Логически все файлы ОС Linux организованы как последовательность байтов. Эта организация распространяется и на операции ввода и вывода. Все вводимые данные представляют собой поток байтов, называемый входным потоком (стандартный ввод). Выводимая информация организована также как последовательный поток данных, направляемых на терминал ‑ стандартный вывод.

Стандартные ввод и вывод могут взаимодействовать с файлами с помощью операции переназначения (переадресации). При этом результаты выполнения любой команды можно передать в файл. Например, после выполнения команды

**ls -l > dir\_file**

будет сформирован файл с именем dir\_file. Этот файл будет содержать полный список файлов, находящихся в текущем каталоге.

С помощью переназначения выхода можно сформировать файл, содержащий произвольную строку. Для этого используется команда echo, которая переадресует свой выход в файл:

**echo my string 1 > my\_file**

При попытке переназначить следующую строку в тот же файл, старое содержимое файла будет уничтожено. Чтобы этого не произошло необходимо при переназначении использовать символы "**>>"**. Так после выполнения команды

echo my string 2 >> my\_file

в файл будет добавлена новая строка.

Если ввести команду cat без параметров, то по умолчанию входным файлом для нее будет считаться стандартный вход, а выходом — стандартный выход. Можно переназначить выход команды в файл на диске cat > file1, таким образом можно получить любой текстовый файл. Для завершения ввода необходимо ввести символ конца файла — **Ctrl+d**.

Аналогично могут быть перенаправлены и входные данные. Для обозначения перенаправления входа используется знак **<**. При этом следует учитывать, что вход данных из файла может использоваться только для тех команд, которые получают данные от стандартного входа. Например, следующая команда обеспечивает подсчет количества строк в файле.

**wc -l < file1**

Операции переназначения можно объединять. Так при выполнении команды

**cat < file1 > file2**

будет произведено копирование содержимого файла file1в файл file2**.**

В отличии от переназначения входов и выходов программные каналы (конвейеры) имеют другую природу. Они являются простым способом организации взаимодействия процессов. Поэтому такой канал перенаправляет данные с выхода одного процесса (в нашем случае команды) на вход другого процесса (команды). Такое перенаправление обозначается символом "**|**". В общем случае конструкция может быть иметь вид

**<команда> <параметры> | <команда> <параметры>**

Так команда

**cat -n my\_file | lpr**

выводит содержимое файла my\_file на печать. Текст печатается с номерами строк.

Использование программных каналов существенно расширяет возможности команды moreи фильтров sort и grep.

Фильтр grep ищет в файлах образец и перечисляет все строки, в которых он есть. Фильтр sort выводит отсортированную версию файла. При этом сам файл не изменяется.

**Порядок выполнения работы**

1. Командой echo -n > имя файла создать файл. С помощью команд **cat** и ls просмотреть его содержимое и длину.
2. Задействовав ряд ключей команды ls***,*** получить результат. Объяснить значение каждого поля.
3. Открыть файл в текстовом редакторе. Набрать несколько осмысленных строк. Завершить работу с редактором.
4. Просмотреть содержимое файла, используя команду cat. Повторно войти в редактор и изменить файл таким образом, чтобы количество строк в нем превышало 25 (число строк на экране). Вновь вывести файл на экран. Объяснить полученные результаты.
5. Используя команду more, добиться поэкранного вывода файла. Объяснить полученные результаты.
6. Командой mkdir <имя каталога> создать каталог. Скопировать туда созданный файл.
7. Попытаться произвести повторное копирование. Объяснить полученные результаты. Используя опции команды cp, добиться результата.
8. Скопировать в каталог тот же файл, указав в качестве приемника имя, отличающееся от старого лишь последним символом. Повторить операцию 4 раза.
9. Командой cd <имя каталога> перейти в созданный каталог. Скопировать в домашний каталог 4 файла, используя разные виды масок (\*, ?, []). Необходимо каждый раз просматривать содержимое каталога командой ls -l. После каждого копирования удалять файлы в каталоге - адресате командой rm, используя необходимые ключи.
10. Удалить все созданные файлы, используя команду rm <имя файла>. Во избежание удаления файлов, созданных другой бригадой, при выполнении этого пункта задания, запрещается пользоваться маской "\*".
11. Создать в рабочем каталоге подкаталог.
12. В соответствии с лабораторным заданием создать пример текстового файла. Файл должен содержать несколько строк информации.
13. В соответствии с вариантом задания создать файл с содержимым каталога. Просмотреть полученный файл.
14. Используя программный канал, обработать результаты выполнения заданной команды соответствующим фильтром. Результат сохранить в файле. команда who выводит список работающих в текущий момент пользователей, команда head выводит первые строки файла. По умолчанию выводятся 10 первых строк. Это значение можно изменить, используя ключ вида -<число строк>.

Варианты заданий

| № | Пункт 13 Операция | Пункт 14 команда фильтр |
| --- | --- | --- |
| 1 | Сокращенный формат содержимого текущего каталога | ls grep |
| 2 | Полная информация по файлам | ls sort |
| 3 | Содержимое каталога рекурсивно вместе с подкаталогами | lsmore |
| 4 | Вывод всех элементов каталога без исключения | catgrep |
| 5 | С указанием размера файла в блоках | cat sort |
| 6 | С добавлением признака каталога (/) после его имени | catmore |
| 7 | Сортировать по размеру блока в обратном порядке | whogrep |
| 8 | С выводом элементов каталога в несколько столбцов | whosort |
| 9 | С сортировкой по времени последней модификации файла | whomore |
| 10 | Сортировать по времени создания файла | head grep |
| 11 | Сортировать по времени создания файла (в обратном порядке) | headsort |
| 12 | С сортировкой по времени последней модификации файла (в обратном порядке) | headmore |

# Лабораторная работа № 3. Работа с файлами

**Цель работы**: изучение особенностей некоторых команд для работы с файлами.

Методические указания

При выполнении работы необходимо использовать следующие команды.

**file [-f<файл>] <имена файлов>**

Команда классификации файлов. При выполнении этой команды система по ключевой информации, имеющейся в файле или по содержимому файла пытается определить тип этого файла. Для текстового файла делается попытка распознать язык программирования. При использовании ключа -f имена файлов извлекаются из файла имя, которого определяется непосредственно за ключом.

**find <путь> <выражение>**

Поиск файла. Первый параметр задает список каталогов, в которых ведется поиск файлов, второй ‑ критерии поиска. В результате выполнения команды выводятся имена файлов для которых значение выражения истинно. Ключи команды можно просмотреть пои вызове страницы справочника man find.

**od [<ключ>] <файл>**

Вывод дампа файла. Формат вывода определяется ключом: восьмеричный (-o), шестнадцатеричный (-x), десятичный (-d) или символьный (-c). Если используется ключ -b, то дамп выводится побайтно в восьмеричной форме.

**cmp [-l] [-s] <файл1> <файл2>**

Команда сравнивает файл1 и файл2. Если эти файлы различаются, то выводятся несовпадающие строки. Команду обычно используют для сравнения бинарных файлов.

-l вывод десятичного номера не совпадающего байта и его восьмеричного значения.

-s возврат только кода завершения: 0 - файлы совпадают, 1 - файлы имеют различия, 2 - сравнение невозможно (файл не найден).

**diff [<ключ>] <файл1> <файл2>**

Команда выводит информацию о несовпадениях файлов. Вывод производится в формате строкового редактора ed.

Файлы в Linux имеют двух владельцев: пользователя и группу. Владельцем-пользователем вновь созданного файла является пользователь, создавший этот файл. Этот пользователь может изменить права доступа к нему по специальной команде.

Права доступа могут изменяться с использованием команды

**chmod <режим> <список файлов>**

Эта команда может задавать правила изменения атрибутов доступа (<режим>) в двух формах ‑ символической и числовой. В символической форме атрибуты задаются в формате

**[who] + | - = [permission]**

Первое поле может содержать один из следующих символов:

a установка атрибутов защиты для всех категорий пользователей (используется по умолчанию);

g установка атрибутов защиты для групп пользователей;

o установка для прочих процессов;

u установка атрибутов только для владельца.

Допустимые операции:

+ добавление прав доступа;

- отмена прав доступа;

= установка перечисленных и отмена остальных прав.

Параметр permissionустанавливает значения атрибутов для перечисленных в первом параметре процессов. Этот параметр может представляться комбинацией из следующих символов:

x ‑ разрешает выполнение (для каталога разрешение поиска);

r ‑ разрешение чтения;

w ‑ разрешение записи;

s ‑ установка пользовательского или группового идентификатора.

Команда chmod g+x-w exampl устанавливает для группы право на выполнение файла exampl и снимет право на запись.

При использовании числовой формы определения атрибутов они задаются в виде восьмеричных кодов, каждая восьмеричная цифра задает атрибуты соответственно для владельца, группы и остальных процессов. Двоичные разряды восьмеричной цифры определяют разрешение ‑ 1 или запрет ‑ 0 последовательно для чтения, записи и исполнения. Например, команда

**chmod 754 my\_file**

дает все права владельцу, разрешает чтение и исполнение для группы и устанавливает право на чтение для всех остальных пользователей.

Исполняемый файл может быть откомпилированной программой или командным файлом интерпретатора shell. В последнем случае должно быть установлено право на чтение, т.к. командный интерпретатор должен иметь возможность считывать команды из файла.

Права доступа для каталогов имеют свои особенности. Право чтения из каталога дает возможность получить только имена находящихся в каталоге файлов. Для получения дополнительной информации о файлах необходимо иметь разрешение на работу с метаданными файла и использованием атрибута исполнение. Право на исполнение необходимо также для того, чтобы сделать каталог текущим. Используя эти свойства можно сделать содержимое каталога невидимым для просмотра, но отдельные файлы из такого каталога будут доступны. Для этого необходимо запретить чтение из каталога, но установить для него атрибут исполнения.

## Лабораторные задания

* 1. Создать текстовый файл содержащий не менее 5 строк текста.
  2. Создать несколько ссылок, просмотреть результаты по команде ls -l. Просмотреть файл, обратившись к нему по ссылке. Удалить одну ссылку и снова просмотреть результаты.
  3. Создать символическую ссылку и просмотреть результат.
  4. Определить тип созданного файла с использованием команды file.
  5. Просмотреть дамп файла. Формат просмотра задается в соответствии с вариантом задания: побайтно — варианты 1, 6, 11; коды символов - 2, 7, 12; десятичные числа - 3, 8; восьмеричные числа - 4, 9; шестнадцатеричные числа - 5, 10.
  6. Скопировать файл в тот же каталог с другим именем. Используя редактор изменить две строки файла-копии.
  7. Выполнить сравнение файлов используя команду cmp ‑ варианты 1, 3, 5, 7, 9, 11; команду diff ‑ варианты 2, 4, 6, 8, 10, 12.

Выполнить поиск файлов по следующим правилам. По числу связей ‑ варианты 1, 7; по имени владельца ‑ варианты 2, 8; по имени группы ‑ варианты 3, 9; по размеру в байтах ‑ варианты 4, 10; по размеру в блоках ‑ варианты 5, 11; по времени модификации ‑ варианты 6, 12.

* 1. В рабочем каталоге создать файл и проанализировать его атрибуты.
  2. Изменить атрибуты каталога и проанализировать результат. Изменение выполнять в соответствии с вариантом.

Чтение для владельца ‑ 1, 4, 7, 10.

Запись для владельца ‑2, 5, 8, 11.

Чтение и запись для владельца ‑ 3, 6, 9, 12.

* 1. Создать каталог и скопировать в него файл. Проанализировать атрибуты этого каталога.
  2. Изменить атрибуты каталога и проанализировать результат.

Чтение для владельца ‑ 1, 6, 11. Запись для владельца ‑ 2, 7. Чтение и запись для владельца ‑ 3, 8. Исполнение для владельца ‑ 4, 9, 12. Чтение и исполнение для владельца ‑ 5, 10.

# Лабораторная работа № 4. Управление процессами

**Цель работы:** ознакомится с некоторыми командами для управления процессами в операционной системе Linux.

## Методические указания

Каждый процесс имеет уникальный идентификатор (**PID)**. Этот идентификатор позволяет системе различать процессы.

Определение значений идентификаторов может быть выполнено с использованием команды

**ps [<ключи>] [<терминал>] [<сортировка>] [<ID>]**

При использовании различных ключей эта команда позволяет получить информацию о процессе.

Примерами ключей команды могут служить:

l длинный формат;

a показывать процессы других пользователей;

e показывать процессы всех пользователей;

r показывать только выполняющиеся процессы.

Полностью значения ключей можно просмотреть с соответствующей странице справочника (следует учитывать, что в некоторых версиях операционной системы значения ключей могут не совпадать).

Параметр txx определяет вывод информации только для процессов, связанных с указанным терминалом. Параметр сортировка задает порядок вывода информации. Указав последний параметр можно вывести данные только для определенных параметром процессов.

Команда

**who [<ключи>]**

выдает список пользователей. Ключи этой команды могут иметь следующие значения:

q выводятся только имена и идентификаторы пользователей;

i, u после имени выводится время;

H выводятся заголовки.

Команда id выводит идентификатор пользователя и группы для текущего процесса.

Одним из способов передачи информации между процессами служат сигналы. Из среды интерпретатора можно отправлять сигналы процессам с помощью команды

**kill [-sig] pid**

посылает процессу сигнал, заданный аргументом sig. В качестве второго аргумента используется идентификатор процесса, который может быть получен по команде ps. Команда kill -l возвращает список символических имен сигналов.

## Лабораторные задания

* 1. Выполнить команду **ps** с различными ключами. Проанализировать результаты. Отразить результаты в отчете по работе.
  2. Выполнить команду **who** с различными ключами. Проанализировать результаты. Отразить результаты в отчете по работе.
  3. Открыть окно второго терминала. Запустить любую программу. Возвратиться на первый терминал и завершить работу программы на втором терминале по команде   
     **kill -9 <идентификатор процесса>**. Проверить результат завершения на втором терминале и по команде **ps**.

# Лабораторная работа № 5. Командные файлы

**Цель работы:** научиться создавать командные файлы с использованием переменных окружения и команды ветвлений.

## Методические указания

Стандартный интерпретатор имеет развитый язык программирования, позволяющий создавать командные файлы (скрипты). Скрипт представляет собой обычный текстовый файл, который включает в себя команды, функции и комментарии. Комментарии начинаются с символа "**#**". Для идентификации типа интерпретатора целесообразно в первой строке файла указывать соответствующий тип, например, **#!/bin/bash**. Командный файл обязательно должен иметь атрибут исполнимый.

Как и в любом языке программирования в командном интерпретаторе существуют переменные. Переменные обозначаются с помощью имен.

Значения присваиваются переменным по оператору присваивания, например, length=80.

Значение переменной можно получить, если перед ее именем указать знак **$**. Так при выполнении команды

**echo $length**

на экран будет выведено **80**. При работе с переменными интерпретатора следует помнить, что значением переменной всегда является строка.

Если переменной присваивается строковое значение, то можно использовать и кавычки и апострофы. Если в строке имеются пробелы, кавычки обязательны, поскольку они позволяют трактовать все символы, включая служебные как элементы строки, а не как специальные управляющие символы. Исключением является символ **$** перед именем переменной, который всегда приводит к ее вычислению. В случае использования апострофа знак доллара не приводит к вычислению значения переменной.

Значения всех установленных переменных можно просмотреть по команде set.

Особое значение имеют обратные апострофы. Они позволяют получать значения переменных как результаты выполнения команд. Если некоторая команда будет заключена в обратные апострофы, то после ее исполнения результаты будут помещены на то место, где эта команда находится.

Важную роль для организации сценариев играют встроенные переменные:

**$1,$2**,... ‑ параметры, задаваемые при запуске сценария;

**$#** ‑ число позиционных параметров;

**$?** ‑ код возврата последнего выполненного процесса;

**$$** ‑ идентификатор текущего shell;

**$!** ‑ идентификатор последнего процесса в фоновом режиме;

**$\*** ‑ все параметры, переданные сценарию (как единая строка);

**$@** ‑ параметры передаются как слова, заключенные в кавычки.

Ветвления в сценарии организуются по команде

**if <команда>**

**then**

**<команды>**

**else**

**<команды>**

**fi**

После ключевого слова if используется специальная команда проверки условий test.

Команда используется в двух формах.

**test <значение> -<опция> <значение>**

или

**test <строка> = <строка>**

Опции задают соответствующие условия. Для различных типов операндов используются разные опции.

Опции для целых операндов:

-gt ‑ больше;

-lt ‑ меньше;

-ge ‑ больше или равно;

-le ‑ меньше или равно;

-eq ‑ равно;

-ne ‑ не равно.

Опции для строк:

-z ‑ строка нулевой длины;

-n ‑ проверка на строковое значение;

=‑ проверка на равенство строк;

!= ‑ проверка на неравенство строк;

-str ‑ проверка на строку ненулевой длины.

При проверке условий можно использовать сложные логические выражения с использованием логических операций. Для представления логических операций используются следующие опции:

-a ‑ логическое И;

-o ‑ логическое ИЛИ;

! ‑ отрицание.

Для работы с файлами существуют специальные опции:

-f ‑ файл существует и является обычным;

-s‑ файл не пустой;

-r ‑ файл доступен для чтения;

-w‑ файл доступен для записи;

-x‑ файл доступен для исполнения;

-d‑ каталог;

-h‑ символьная ссылка.

Вместо ключевого слова test в команде проверки условия можно использовать квадратные скобки, которые отделяются от условия пробелом, табуляцией или символом перехода на новую строку. Два следующих выражения будут эквивалентными:

**test $num -eq 10**

**[ $num -eq 10 ]**

Оператор множественного выбора **case** имеет следующий синтаксис:

**case строка in**

**шаблон) список команд;;**

**шаблон) список команд;;**

**...**

**esac**

Данный оператор в основном тождественен широко применяемой в языке C конструкции switch. Однако в отличии от последней здесь не требуется явным образом использовать оператор break - в случае совпадения одного из условий с образцом дальнейший перебор вариантов прекращается автоматически.

Если необходимо определить действие по умолчанию, то есть на тот случай, если строка не совпала ни с одним из шаблонов, то в поле образца необходимо поставить символ "**\***".

Одним из наиболее мощных средств автоматизации выполнения операций в shell является оператор цикла с перечислением for, который в общем виде можно записать следующим образом:

**for переменная [in список\_значений]**

**do**

**список команд**

**done**

Здесь переменная последовательно принимает значения из списка. Если список значений переменной отсутствует, то при организации цикла используются параметры командной строки при вызове скрипта.

Особого упоминания заслуживает конструкция for <переменная> in \* которая обозначает "все файлы и подкаталоги внутри данного каталога". Она широко используется, например, при анализе содержимого требуемого каталога.

## Лабораторные задания

1. Подготовить и выполнить сценарий, содержащий установку и отображение значений переменных.

2. В соответствии с вариантом задания подготовить и выполнить сценарий, использующий условие.

2.1. Проверить существует ли в текущем каталоге файл и вывести на экран его содержимое, имя файла определяется через параметр. Если файл не существует, сформировать файл с помощью команды **cat**.

2.2. Определить количество работающих в данный момент пользователей. Если число пользователей больше 10 выдать сообщение.

2.3. Вывести на экран содержимое файла. Если файла в текущем каталоге нет, скопировать файл из другого каталога. Имя файла передается через параметр.

2.4. Прочитать содержимое указанного каталога в файл. Если каталог пуст, выдать на экран сообщение. Имя каталога вводится с клавиатуры.

2.5. Просмотреть содержимое текущего каталога, ввести имя одного из файлов. Если этот файл имеет ненулевую длину, то вывести его содержимое на экран.

2.6. Если указанный в параметре файл не имеет установленного атрибута разрешения, то необходимо установить этот параметр.

2.7. Проверить существует ли в текущем каталоге файл, имя файла вводится с клавиатуры. Если файл не существует, сформировать его.

2.8. Определить количество файлов с расширением **koi**, находящихся в каталоге **/home/student**. Если таких файлов нет, то выдать на экран сообщение.

2.9. Прочитать содержимое указанного каталога в файл. Если каталог не пуст, выдать на экран сообщение. Имя каталога передается через параметр.

2.10. Вывести на экран содержимое файла. Если файла в текущем каталоге нет, скопировать файл из другого каталога. Имя файла вводится с клавиатуры. Имя каталога передается через параметр.

2.11. Определить количество работающих в настоящее время пользователей с именем **student**. Если количество пользователей больше числа передаваемого в сценарий через параметр, вывести сообщение.

2.12. Проверить является ли указанный в параметре файл каталогом. Вывести соответствующую информацию на экран. Если это каталог, то установить разрешение записи в этот каталог.

3. Используя оператор множественного выбора case, разработать и отладить сценарий на языке shell, реализующий меню типа

Введите нужный пункт меню:

1. Операция 1

2. Операция 2

Имена файлов, образцы для фильтров должны передаваться в качестве параметров. Анализируемые в меню значения вводятся с клавиатуры.

### Варианты заданий

|  |  |
| --- | --- |
| Операция 1: | Произвести поиск всех файлов, которые изменялись столько дней назад, каков номер варианта. Поиск ведется от домашнего каталога. При этом не указывать полный путь. Имена файлов приемников результата и потока ошибок принимаются соответствующими номеру варианта с префиксами **out** и **err** соответственно. Обоим файлам присвоить расширение, соответствующее:  (1,4,9,12) - номеру процесса,  (2,6,8,11) - номеру последнего фонового процесса,  (3,5,7,10) - номеру варианта. |
| Операция 2: | Произвести поиск всех процессов, принадлежащих (варианты 1,3,4,7,9,12) или нет (2,5,6,8,10,11) конкретному пользователю, зарегистрированному в системе. Имя пользователя, как и имя файла-приемника, вводится с клавиатуры. Вывод команды должен быть отсортирован в алфавитном порядке. У файла-приемника должно быть расширение, соответствующее введенному идентификатору пользователя. |

4. Перед выполнением задания создать в своем каталоге набор файлов с расширениями **\*.с, \*.cpp, \*.h и \*.txt**.

4.1. Используя цикл for, распечатать из текущего каталога содержимое всех файлов с расширениями **\*.c** и **\*.cpp**, в именах которых присутствуют цифры. Перед содержимым каждого файла напечатать его имя.

4.2. Используя цикл **for**, создать в каталоге "**./links**" символические ссылки на все файлы текущего каталога с добавлением к имени файла "**.link**".

4.3. Синхронизировать содержимое каталогов "**./**" и "**./backup**" путем создания символических ссылок на недостающие файлы.

4.4. Вывести для определенных каталогов имена текстовых файлов, для которых разрешена запись. Имена каталогов задаются через параметры.

4.5. Проверить существует ли в каталогах, заданных через параметры при вызове сценарии, файл. Имя файла вводится с клавиатуры.

4.6. Проверить работают ли в настоящее время в системе пользователи, имена которых заданы списком параметров.

4.7. Вывести для каталога (имя каталога вводится с клавиатуры) список файлов, для которых разрешены исполнение и чтение.

4.8. Создать резервные копии текстовых файлов, имеющих атрибут разрешения для записи.

4.9. Копировать в каталог, имя которого вводится с клавиатуры, файлы, у которых имя начинается с букв "a" или "z", если эти файлы не являются каталогами.

4.10. Проверить имеются ли в каталогах, имена которых определяются параметрами сценария, файлы, являющиеся исходными текстами программ.

4.11. Проверить установлены ли атрибуты разрешения чтения и исполнения для файлов, имена которых перечисляются в списке параметров при вызове сценария.

4.12. Преобразовать имена файлов с расширением **h** в имена с расширением **hpp**. Каталоги, в которых выполняются преобразования, задаются через параметры.

# Часть 2. Технология разработки программ в среде Linux

## Методические указания

В операционной системе Linux для компиляции программ написанных на языке C\C***++*** используется компилятор GNU C\C++.

Для компиляции C программы, состоящей из одного файла используется компилятор **cc** или **gcc**, а для компиляции программы, написанной на C++ используется компилятор **g++.**

Исторически сложилось так, что если задать строку компиляции cc proba.c, то результатом будет исполняемый файл **a.out**. Это имя подставляется автоматически, если явно не указано имя выходного файла. Для задания имени выходного файла используется опция ***–***o <имя файла>. При этом тип файла выбирается автоматически. Если перед именем компилируемого файла стоит –c, то создается объектный выходной файл с расширением .o, в противном случае создается исполняемый файл.

Вызов cc также используется при необходимости собрать исполняемый файл из нескольких объектных. Для этого необходимо вызвать его следующим образом:

**cc <объектные модули> -o <имя исполняемого файла>**

Если программа использует дополнительные библиотеки – возникает необходимость указания дополнительных путей поиска заголовочных и библиотечных файлов. Для этого используются опции –I<путь> и –L<путь>. При необходимости могут присутствовать несколько опций –I и –L.

# Лабораторная работа № 6. Файловые API

**Цель работы**: изучить использование вызовов системных функций для работы с файлами.

## Методические указания

Для работы с файлами следует использовать следующие функции API.

**#include <sys/types.h>**

**#include <sys/stat.h>**

**#include <fcntl.h>**

**int open(const char \*path, int flags, mode\_t mode);**

Первый параметр задает имя файла, второй параметр показывает, какие виды доступа к файлу разрешены вызывающему процессу. Этот параметр может принимать следующие значения:

O\_RDONLY ‑ открытие файла только для чтения;

O\_WRONLY ‑ открытие файла только для записи;

O\_RDWR ‑ открытие файла для чтения и записи.

Значение параметра может логически складываться с модификаторами:

O\_APPEND ‑ данные добавляются в конец файла;

O\_CREAT ‑ создается файл, если он не существует;

O\_TRUNC ‑ если файл существует, то его содержимое теряется, а размер устанавливается равным 0;

O\_EXCL ‑ используется совместно с флагом O\_CREAT, в этом случае попытка создать файл, если он уже существует оканчивается неудачей.

Третий параметр необходим только при создании нового файла, обычно он задается в виде восьмеричной константы и определяет права доступа к этому файлу.

После успешного открытия файла функция возвращает значение дескриптора файла.

Чтение данных выполняется с использованием функций из библиотеки языка Си. В частности, для чтения можно использовать функцию:

**int read(int fdes, char \*buf, size\_t count);**

Запись в файл может выполняться по функции:

**int write(int fdes, char \*buf, size\_t count);**

В качестве первого параметра используется дескриптор файла. Второй параметр указывает на буфер обмена. Третий параметр ‑ длина буфера.

Закрывается файл функцией

**int close(int fdes);**

аргументом функции является дескриптор соответствующего файла.

При выполнении работы необходимо использовать функции для работы с файлами и каталогами, определенные стандартами UNIX и POSIX. Результаты выполнения функций должны проверяться по возвращаемым значениям. В случае возникновения ошибки необходимо выводить соответствующие сообщения с использованием стандартных функций вывода или специальных функций UNIX для вывода сообщений ‑ strerror() и perror().

В процессе выполнения программы на терминал должны выдаваться сообщения о всех фазах ее работы и об основных состояниях файлов и каталогов. Эти состояния могут быть получены при использовании функций stat(), fstat() или access().

Создание и уничтожение жестких ссылок на файлы организуются при использовании функций link() и unlink(). Изменения режимов доступа к файлам выполняются функциями chmod() и fchmod().

Создание и уничтожение каталогов производится функциями: mkdir()rmdir(). Для присмотра файла каталога он должен быть открыт с помощью специальной функции opendir(). Закрытие каталога выполняется функцией closedir(). Для присмотра каталога аналогично управляющей структуре **FILE**, применяемой при работе с обычными файлами используется структура DIR, с помощью которой организуется доступ к файлу каталога. Чтение очередной записи каталога выполняется функцией readdir(). Для удобства работы с каталогом могут также использоваться функции установки указателя текущей записи в начало каталога rewinddir(), определения текущей позиции указателя чтения каталога telldir() и перемещения этого указателя в заданную позицию seekdir().

## Лабораторные задания

1. В соответствии с вариантом задания разработать и отладить программу. Исходные данные вводятся с клавиатуры и записываются в текстовый файл. Программа читает эти данные, после обработки результаты также помещаются в файле.

1.1. Из текста удалить четвертое слово.

1.2. Сформировать файл, содержащий записи по результатам сдачи очередного экзамена студентами группы. Из файла выбрать записи для студентов, получивших отличные оценки и записать их в новый файл.

1.3. В тексте добавить после третьего слова новое слово.

1.4. Сформировать файл, содержащий записи по результатам сдачи очередного экзамена студентами группы. Сгруппировать записи по оценкам.

1.5. В тексте удалить лишние пробелы.

1.6. В тексте имеются произвольно расположенные русские и английские слова. Разделить текст на два файла, в одном должны находиться английские слова, в другом ‑ русские.

1.7. Для заданного текста определить длину содержащейся в нем максимальной серии символов, отличных от букв.

1.8. Сформировать файл содержащий заключенные в круглые скобки последовательности символов исходного текста.

1.9. Имеется текст со сведениями о сотрудниках предприятия, содержащими год рождения. Выбрать и записать в файл записи для сотрудников младше заданного возраста.

1.10. Из файла, содержащего сведения о студентах сформировать файл, в который входят только фамилии.

1.11. Из текста выбрать четные слова.

1.12. В тексте поменять местами первое и последнее слова.

2. В соответствии с вариантом задания разработать и отладить программу для анализа состояния каталога.

2.1. Определить количество файлов с указанным расширением, находящихся в заданном каталоге. Если таких файлов нет, то выдать на экран сообщение. Имя каталога и расширение передаются в программу через параметры командной строки.

2.2. Прочитать содержимое указанного каталога в файл. Если каталог пуст, выдать на экран сообщение. Имя каталога вводится с клавиатуры.

2.3. Просмотреть содержимое текущего каталога, ввести с клавиатуры имя одного из файлов. Если этот файл имеет ненулевую длину, то вывести его содержимое на экран.

2.4. Если указанный в параметре командной строки файл не имеет установленного атрибута разрешения для выполнения, то необходимо установить этот параметр.

2.5. Проверить является ли указанный в параметре файл каталогом. Вывести соответствующую информацию на экран. Если это каталог, то установить разрешение записи в этот каталог.

2.6. Вывести для определенного каталога имена текстовых файлов, для которых разрешена запись. Имя каталога задается через параметр командной строки.

2.7. Вывести для каталога (имя каталога вводится с клавиатуры) список файлов, для которых разрешены исполнение и чтение.

2.8. Создать резервные копии текстовых файлов, имеющих атрибут разрешения для записи.

2.9. Прочитать содержимое указанного каталога в файл. Если каталог не пуст, выдать на экран сообщение. Имя каталога передается через параметр командной строки.

2.10. Распечатать из текущего каталога содержащие цифры имена всех файлов с расширениями \*.c и \*.cpp.

2.11. Создать в каталоге "./links" символические ссылки на все файлы текущего каталога с добавлением к имени файла ".link".

2.12. Копировать в каталог, имя которого вводится с клавиатуры, файлы, у которых имя начинается с букв "a" или "z", если эти файлы не являются каталогами.

# Лабораторная работа № 7. Создание процессов

**Цель работы**: освоить основные приемы работы с параллельными процессами.

## Методические указания к выполнению работы

Новый процесс создается системным вызовом fork(). При этом порождаемый процесс - потомок является точной копией процесса - родителя. Они различаются тем, что потомок имеет отличные от родительского процесса идентификаторы (**PID** и **PPID**). Поскольку порожденный процесс имеет одинаковый с родительским процессом программный код, для различия в алгоритмах выполнения можно использовать код возвращаемый функцией fork(). Для родительского процесса при нормальном завершении возвращается идентификатор порожденного процесса, процесс - потомок получает от fork() код возврата, равный нулю. При неудачном завершении возвращается код ошибки равный **-1** и устанавливается значение **errno**.

Для того чтобы порожденный процесс выполнял независимые от процесса - родителя действия в нем выполняется системный вызов exec(), по которому запускается другая программа.

Синхронизация процесса - родителя и процесса - потомка выполняется по системному вызову wait(). Для завершения процессов служит функция \_exit().

Простейшая работа с таймером организуется с помощью функции

**#include <unistd.h>**

**long alarm(long alarm);**

Эта функция посылает процессу сигнал **SIGALRM** по истечении интервала времени, заданного параметром функции. Интервал задается в секундах. Обработка сигнала выполняется по общим правилам. Функция возвращает значение оставшегося времени до срабатывания таймера, установленного при предыдущем вызове функции.

Более мощный механизм реализован через функции работы с интервальными таймерами.

**#include <sys/time.h>**

**getitimer(int which, struct itimerval \*value);**

**setitimer(int which, const struct itimerval \*value,  
 struct itimerval \*ovalue);**

Первый аргумент функции задает тип таймера. Он может принимать одно из трех значений:

**ITIMER\_REAL** ‑ таймер реального времени, по истечении заданного времени генерируется сигнал **SIGALRM**.

**ITIMER\_VIRTUAL** ‑ таймер изменяется только когда процесс выполняется в режиме задачи, генерируется сигнал **SIGVTALRM**

**ITIMER\_PROF** ‑ таймер изменяется, когда процесс выполняется в режиме ядара или задачи, генерируется сигнал **SIGPROF**.

Второй аргумент адресует структуру, в которой хранятся параметры таймера.

**struct itimerval {**

**struct timeval it\_interval; // интервал таймера**

**struct timeval it\_value; // текущее значение**

**};**

Структура timeval определена следующим образом:

**struct timeval {**

**long tv\_sec; // время в секундах**

**long tv\_usec; // время в микросекундах**

**};**

Третий аргумент функции setitimer служит для возврата предыдущее значение таймера, если он установлен в 0, то старое значение таймера не возвращается.

Для работы с сигналами в операционных системах семейства UNIX используется функция, обеспечивающие ненадежные сигналы (signal), и современный вариант API, включающий в себя группу функций управления сигналами.

Модель надежных сигналов, реализованная в интерфейсе POSIX.1 основана на понятии набора сигналов, описываемого переменной типа sigset\_t. Каждый бит этой переменной связан с определенным сигналом, поэтому число сигналов ограничено значением 32. Обработка сигнала в этом случае выполняется функцией с прототипом

**void (\*sigset(int sig\_num,void (\*handler)int)))(int);**

Для работы с сигналами в этом случае используется набор функций:

**#include <signal.h>**

**int sigemptyset(sigset\_t mask);**

**int sigaddset(sigset\_t mask, const int sig\_num);**

**int sigdelset(sigset\_t mask, const int sig\_num);**

**int sigfillset(sigset\_t mask);**

**int sigismember(const sigset\_t\* mask, const int sig\_num);**

Параметр mask является сигнальной маской и определяет какие сигналы из посылаемых процессу блокируются. Все перечисленные функции работают с сигналами, перечисленными в маске. Так функция sigemptyset сбрасывает все биты, соответствующие сигналам в маске. Функции sigaddset и sigdelset устанавливают и сбрасывают соответствующие сигналы. Функция sigfillset устанавливает все сигналы. Последняя функция sigismember проверяет установлен ли сигнал в маске.

В последних версиях ОС используется интерфейс прикладного программирования sigaction.

Вся необходимая для управления сигналами информация передается через структуру sigaction, имеющую поля:

void (\*sa\_handler)() ‑ указатель на обработчик сигнала;

void (\*sa\_sigaction)(int, siginfo\_t, void \*) ‑ указатель на обработчик сигнала при установленном флаге **SA\_SIGINFO**;

sigset\_t sa\_mask ‑ маска сигналов;

int sa\_flags ‑ флаги.

## Лабораторные задания

1. Разработать программу, выполняющую "разветвление" посредством системного вызова fork(). Вывести на экран идентификаторы **PID** и **PPID** для родительского и дочернего процессов. Разработать функцию перенаправления стандартного вывода в файл.

1.1. Приостановить на 1 с родительский процесс. В дочернем с помощью системного вызова system() выполнить стандартную команду ps, перенаправив вывод в файл номер 1. Вслед за этим завершить дочерний процесс. В родительском процессе вызвать **ps** и перенаправить в файл номер 2. Освободить ячейку таблицы процессов порожденного процесса.

1.2. Приостановить на 1 с родительский процесс. Выполнить в дочернем процессе один из системных вызовов exec(), передав ему в качестве параметра стандартную программу **ps**. Аналогично выполнить вызов **ps** в родительском процессе. Результаты работы команд **ps** в обоих процессах перенаправить в один и тот же файл. Освободить ячейку таблицы процессов порожденного процесса.

1.3. В самом начале программы, до порождения дочернего процесса, сделать процесс лидером группы и сеанса. Вызвать в обоих процессах с помощью любого системного вызова стандартную команду **ps** и перенаправить ее выход в файл. Модифицировать программу таким образом, чтобы лидером сеанса и группы стал родительский процесс. Повторно запустить программу, предварительно сохранив дампы предыдущего запуска. Объяснить произошедшие изменения. Освободить ячейку таблицы процессов порожденного процесса.

1.4. Определить в программе глобальную переменную **var** со значением, равным 1. Переопределить стандартный вывод и родительского, и дочернего процессов в один и тот же файл. До выполнения разветвления инкрементировать переменную **var**, причем вывести ее значение, как до увеличения, так и после. В родительском процессе увеличить значение переменной на 3, а в дочернем на 5. Вывести значение переменной до увеличения и после него внутри каждого из процессов. Результат пояснить.

1.5 Приостановить на 1 с дочерний процесс. В дочернем процессе с помощью системного вызова system() выполнить стандартную команду **ps**, перенаправив вывод в файл номер 1. Вслед за этим завершить дочерний процесс. В родительском процессе вызвать **ps** и перенаправить в файл номер 2. Освободить ячейку таблицы процессов порожденного процесса.

1.6. Приостановить на 1 с дочерний процесс. Выполнить в дочернем процессе один из системных вызовов exec(), передав ему в качестве параметра стандартную программу **ps**. Аналогично выполнить вызов **ps** в родительском процессе. Результаты работы команд **ps** в обоих процессах перенаправить в один и тот же файл. Освободить ячейку таблицы процессов порожденного процесса.

1.7. В самом начале программы, до порождения дочернего процесса, сделать процесс лидером группы и сеанса. Вызвать в обоих процессах с помощью любого системного вызова стандартную команду **ps** и перенаправить ее выход в файл. Модифицировать программу таким образом, чтобы лидером сеанса и группы стал дочерний процесс. Повторно запустить программу, предварительно сохранив дампы предыдущего запуска. Объяснить произошедшие изменения. Освободить ячейку таблицы процессов порожденного процесса.

1.8. Программа порождает через каждые 2 секунды 5 новых процессов. Каждый из этих процессов выполняется заданное время и останавливается, сообщая об этом родителю. Программа-родитель выводит на экран все сообщения об изменениях в процессах.

1.9. Программа запускает с помощью функции exec() новый процесс. Завершить процесс-потомок раньше формирования родителем вызова. Повторить запуск программы при условии, что процесс потомок завершается после формирования вызова wait(). Проанализировать результаты.

2. Таймеры.

2.1. Программа генерации сообщений пользователю с заданным интервалом времени. Продолжение работы программы определяется сигналом с терминала.

2.2. Программа выполняет работу в бесконечном цикле. Каждые 5 секунд производится прерывание цикла и на терминал выдается запрос на подтверждение продолжения работы.

2.3. Программа приостанавливается, а затем по истечению заданного времени продолжает свою работу, периодически выдавая на терминал сообщения о прошедшем после последней приостановки времени.

2.4. Программа генерации сообщений о том, что она была активна в течении определенного времени.

2.5. Вывести на экран значения реального, виртуального и общего времени, в течение которого работала программа пользователя. Под общим временем понимается время, затраченное на решение пользовательских и системных задач.

2.6. Вывести на экран значение времени, затраченного на решение системных задач (перепланирование и т.д.) во время обработки задачи пользователя.

2.7. Проверить, сколько микросекунд прошло с момента установки таймера функцией alarm() до получения соответствующего сигнала. Принять аргумент функции alarm() равным целому числу, лежащему в диапазоне от 2 до 6 секунд.

2.8. Реализовать программу, которая через заданные промежутки времени выводит на экран текущий список процессов. Воспользоваться вызовом alarm.

2.9. Написать программу, которая будет интерпретировать свой второй параметр как команду, во время заданное через первый параметр.

2.10. Написать программу, которая при получении сигнала **SIGUSR2** будет начинать обратный отсчет со случайной величины в диапазоне от 15 до 100, где каждая единица соответствует 1 секунде. Производить выдачу текущего значения каждые 10 секунд, а последние 10 значений – каждую секунду. По окончании отсчета программа должна переходить в начальное состояние.

2.11. Программа запускает новый процесс после истечения заданного времени работы в режиме задачи.

2.12. Программа формирует сообщение в том случае, когда ее процесс - потомок просуществует заданное время.

3. Сигналы.

3.1. Два процесса синхронизируют свою работу путем обмена сигналами.

3.2. Проверить, есть ли в сигнальной маске процесса сигнал **SIGINT**. Если он отсутствует, установить его, в противном случае снять.

3.3 Путем вызова fork() породить 5-7 дочерних процессов, предварительно установив в родительском процессе свой собственный обработчик сигнала **SIGCHLD**. Каждый раз при завершении процесса - потомка выдавать на экран сообщение с указанием **PID** и **PPID** завершившегося процесса. Родительский и дочерние процессы должны функционировать параллельно.

3.4 Путем вызова fork() породить 5-7 дочерних процессов, предварительно установив в родительском процессе свой собственный обработчик сигнала **SIGUSR2**. Каждый раз при завершении процесса - потомка посылать сигнал родительскому процессу и выдавать на экран сообщение с указанием **PID** и **PPID** завершившегося процесса. Родительский и дочерние процессы должны функционировать параллельно.

3.5. Получить сигнальную маску процесса и вывести в файл протокола существующий набор сигналов. Заменить набор сигналов на противоположный (инверсный) и вновь занести полученные значения в файл протокола.

3.6. Установить собственные обработчики для сигналов **SIGINT**, **SIGUSR1** и **SIGQUIT**. Приостановить текущий процесс. В случае посылки из командной строки от другой консоли сигналов **SIGINT** и **SIGUSR1** выдавать сообщения об этом, а в случае получения сигнала **SIGQUIT** завершить работу, выдав сообщение. Сообщения выводятся из обработчиков сигналов.

3.7. Установить собственные обработчики для сигналов **SIGINT**, **SIGUSR1** и **SIGKILL**. Получить сигнальную маску процесса и запретить реагирование на эти сигналы. С другой виртуальной консоли поочередно послать в этот процесс указанные сигналы. После получения соответствующего сигнала выдавать на экран сообщение. Объяснить полученный результат.

3.8. Процесс настраивается на выполнение одной из двух операций в соответствии с сигналом, формируемым пользователем с терминала.

3.9. Написать программу, вводящую данные со стандартного потока ввода и по приходу сигнала **SIGINT** сбрасывающую результаты в файл.

3.10. Написать программу, создающую несколько своих копий, которые реализуют задержку на случайный промежуток времени, а затем завершается и как код возврата передает свой **PID**. Родитель должен выводить **PID** потомков при их запуске и при их завершении. Обработку завершения потомков реализовать через **SIGCHLD**.

3.11. Написать программу, которая при получении сигнала **SIGUSR1** будет читать строку из определенного файла, которая в свою очередь является полным путем к файлу, и будет выводить на экран его содержимое.

3.12. Написать программу, которая посылает сигнал, код которого задан третьим параметром командной строки, на **PID**, заданный вторым параметром во время, заданное первым параметром.

# Лабораторная работа № 8. Каналы

**Цель работы**: научится создавать и использовать для обмена информацией между процессами каналы.

## Методические указания

Создание каналов выполняется с использованием функции

**#include <unistd.h>**

**int pipe(int \*filedes);**

Функция возвращает два дескриптора:

filedes[0] - для записи;

filedes[1] - для чтения.

Обмен информацией выполняется с использование функций записи и чтения API. Каналы используются для родственных процессов.

Независимые процессы могут использовать именованные каналы. Такие каналы создаются функцией

**#include <sys/types.h>**

**#include <sys/stat.h>**

**#include <sys/fcntl.h>**

**#include <unistd.h>**

**int mknod(const char \*pathname, mode\_t, dev\_t dev);**

Первый параметр специфицирует имя создаваемого канала, параметр mode задает права доступа и тип (для именованного канала используется значение **S\_IFIFO**. Третий параметр игнорируется. Функция возвращает признак нормального завершения ‑ 0, при ошибке возвращается значение -1.

Уничтожение канала выполняется по функции

**int unlink(const char \*pathname)**

## Лабораторные задания

Написать две программы, которые создают между собой канал. Одна программы выполняет роль клиента, вторая служит сервером. Функции клиента и сервера определяются вариантами заданий на выполнение лабораторной работы. В четных вариантах задания использовать именованные каналы.

### Варианты заданий

1 Клиент передает серверу через канал запрос в виде полного пути к файлу. Сервер читает этот файл и передает клиенту его содержимое или сообщение об ошибке, если файл с указанным именем не существует или не доступен для чтения. Клиент выводит принятые данные на терминал.

2 Клиент и сервер обмениваются сообщениями, вводимыми с клавиатуры. Программы запускаются на разных терминалах. Принятые сообщения выводятся на экран.

3 Сервер выполняет команду ps и результаты ее выполнения передаются клиенту, который выводит их на терминал.

4 Клиент и сервер обмениваются между собой сообщениями. Программы запускаются на разных терминалах. Каждая программа записывает принятые сообщения в файл, расширение которого является значением идентификатора процесса, соответствующего данной программе.

5 Клиент передает серверу запрос в виде полного пути к файлу. Сервер читает этот файл и передает клиенту его содержимое или сообщение об ошибке, если файл не существует или не доступен для чтения. Клиент записывает полученную информацию в файл в текущем каталоге с тем же именем и дополняет его расширением result.

6 Клиент принимает с клавиатуры команды и передает их серверу. Сервер выполняет принятые команды и возвращает результаты их выполнения клиенту. Принимаемые данные клиент выводит на терминал. Программы запускать на разных терминалах.

7 Клиент принимает с клавиатуры команды и передает их серверу. Сервер выполняет эти команды, результаты возвращаются клиенту, который записывает их в файл.

8 Клиент запрашивает у сервера количество файлов, находящихся в указанном каталоге. Полученный результат выводится клиентом на терминал.

9 Клиент формирует запрос содержащий имя файла. Сервер определяет является ли указанный файл каталогом и формирует соответствующий ответ. Ответ выводится клиентом на экран.

10.Клиент формирует запрос, содержащий имя каталога. Сервер просматривает каталог и передает клиенту количество подкаталогов, имеющих в данном каталоге. Клиент выводит полученную информацию на экран.

11. Клиент формирует запрос, содержащий имя каталога. Сервер проверяет имеется ли разрешение записи в этот каталог, при необходимости устанавливает это право и информирует клиента о результатах выполнения операции. Клиент выводит на экран полученное от сервера сообщение.

12. Клиент запрашивает у сервера количество работающих в данный момент времени пользователей. Если количество пользователей больше заданного числа на терминал выводится сообщение.

# Лабораторная работа № 9. Сообщения

**Цель работы**: научится организовывать обмен данными между процессами с использованием сообщений.

## Методические указания

Для идентификации сообщений можно использовать ключи, которые генерируются в системе при вызове функции

**key\_t ftok(char \*filename, char proj);**

В качестве имени файла можно задавать имя любого существующего файла, в частности, для определенности можно использовать имя самой программы.

Для обмена используются очереди сообщений. Очередь создается функцией

**#include <sys/types.h>**

**#include <sys/ipc.h>**

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

**int msgget(key\_t key, int msgflg);**

Если процессу необходимо создать новую очередь сообщений, то флаг должен содержать макрос **IPC\_CREAT**, а также права на чтение и запись сообщений в очередь (**0644**). При нормальном завершении функция возвращает идентификатор очереди, в случае ошибки возвращается значение -1.

Посылка и прием сообщений организуются при вызове функций

**int msgsnd(int msgid, struct msgbuf \*msgp, int msgsz, int msgflg);**

и

**int msgrcv(int msgid, struct msgbuf \*msgp, int msgsz, long msgtyp, int msgflg);**

Первый параметр задает идентификатор очереди. Второй параметр является указателем на сообщение. Сообщение представляет собой структуру

**struct msgbuf {**

**long mtype; /\* тип сообщения \*/**

**char mtext[]; /\* указатель на буфер сообщения \*/**

**};**

Параметр **msgsz** определяет длину сообщения. При значении параметра **msgflg=0** процесс может блокироваться до тех пор пока функция не будет выполнена. Параметр **msgtyp** задает правила выбора сообщения из очереди. При нулевом значении параметра из очереди извлекается самое старое сообщение любого типа. Положительное значение определяет прием самого старого сообщения указанного типа.

Удаление очереди из системы производится при вызове функции

**int msgctl(int msgid, int cmd, struct msgbuf \*msgp);**

при значении параметра cmd равном **IPC\_RMID**.

## Лабораторные задания

Написать две программы, одна из которых выполняет роль клиента, вторая служит сервером. Клиент и сервер обмениваются между собой сообщениями. Функции клиента и сервера определяются вариантами заданий на выполнение лабораторной работы.

### Варианты заданий

* 1. Клиент принимает с клавиатуры команды и передает их серверу. Сервер выполняет принятые команды и возвращает результаты их выполнения клиенту. Принимаемые данные клиент выводит на терминал. Программы запускать на разных терминалах.
  2. Клиент запрашивает у сервера количество работающих в данный момент времени пользователей. Если количество пользователей больше заданного числа на терминал выводится сообщение.
  3. Клиент и сервер обмениваются между собой сообщениями. Программы запускаются на разных терминалах. Каждая программа записывает принятые сообщения в файл, расширение которого является значением идентификатора процесса, соответствующего данной программе.
  4. Клиент передает серверу через канал запрос в виде полного пути к файлу. Сервер читает этот файл и передает клиенту его содержимое или сообщение об ошибке, если файл с указанным именем не существует или не доступен для чтения. Клиент выводит принятые данные на терминал.
  5. Сервер выполняет команду ps и результаты ее выполнения передаются клиенту, который выводит их на терминал.
  6. Клиент формирует запрос, содержащий имя файла. Сервер определяет является ли указанный файл каталогом и формирует соответствующий ответ. Ответ выводится клиентом на экран.
  7. Клиент формирует серверу запрос, содержащий имя каталога. Сервер проверяет, имеется ли разрешение записи в этот каталог, при необходимости устанавливает это право и информирует клиента о результатах выполнения операции. Клиент выводит на экран полученное от сервера сообщение.
  8. Клиент передает серверу запрос в виде полного пути к файлу. Сервер читает этот файл и передает клиенту его содержимое или сообщение об ошибке, если файл не существует или не доступен для чтения. Клиент записывает полученную информацию в файл в текущем каталоге с тем же именем и дополняет его расширением result.
  9. Клиент формирует запрос, содержащий имя каталога. Сервер просматривает каталог и передает клиенту количество подкаталогов, имеющих в данном каталоге. Клиент выводит полученную информацию на экран.
  10. Клиент принимает с клавиатуры команды и передает их серверу. Сервер выполняет эти команды, результаты возвращаются клиенту, который записывает их в файл.
  11. Клиент запрашивает у сервера количество файлов, находящихся в указанном каталоге. Полученный результат выводится клиентом на терминал.
  12. Клиент и сервер обмениваются сообщениями, вводимыми с клавиатуры. Программы запускаются на разных терминалах. Принятые сообщения выводятся на экран.

# Библиографический список

* 1. Гласс Г., Эйбле К. UNIX для программистов и пользователей. – СПб.: БХВ-Петербург, 2004.
  2. Рейчард К., Фолькердинг П. Linux: справочник. ‑ СПб.: Питер Кон, 1999. ‑ 480 с.
  3. Робачевский А.М. Операционная система UNIX. ‑ СПб.: BHV-Санкт-Петербург, 1997. ‑ 528 с.
  4. Теренс Чан Системное программирование на С++ для UNIX. К.: Издательская группа BHV, 1997. ‑ 592 с.
  5. Фуско Дж. Linux. Руководство программиста. – СПб.: Питер, 2011.
  6. Хэвиленд К., Грэй Д., Салама Б. Системное программирование в UNIX. Руководство программиста. – М., ДМК Пресс, 2000.