Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

Пермский национальный исследовательский политехнический университет

Кузнецов Д. Б.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

по дисциплине

СИСТЕМНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

Пермь, 2016

Литература основная

1. Кузнецов Д.Б. Операционные системы: конспект лекций для студентов заочного отделения.- На электронном носителе, находится в издании.
2. Робачевский А.М. Операционная система UNIX: Учеб. пособие для вузов.- СПБ: BHV-СПБ, 2000.- 514с.

Литература дополнительная

1. Келли-Бутл С. Введение в UNIX: Пер. с англ.- М.: Лори, 1997.- 341с.
3. Рейчард К., Фостер-Джонсон Э. UNIX.- СПБ: Питер, 1999.- 374с.

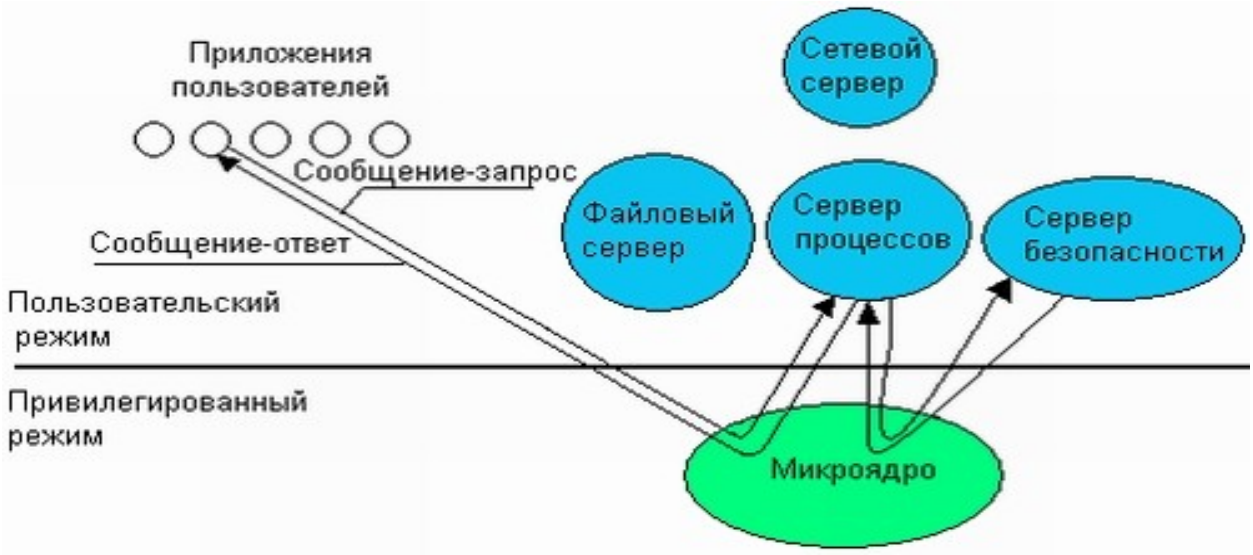
Лабораторная работа 1

Основное содержание работы

Знакомство с операционной системой Minix. Знакомство с процессом установки в виртуальной машине. Подготовка к работе.

Краткие теоретические сведения

ОС Minix относится к микроядерным операционным системам. В отличие от традиционной архитектуры, согласно которой операционная система представляет собой монолитное ядро, реализующие основные функции по управлению аппаратными ресурсами и организующее среду для выполнения пользовательских процессов, микроядерная архитектура распределяет функции ОС между микроядром и входящими в состав ОС системными сервисами, реализованными в виде процессов, равноправных с пользовательскими приложениями.



Микроядро реализует базовые функции операционной системы, на которые опираются эти системные сервисы и приложения. В результате, такие важные компоненты ОС как файловая система, сетевая поддержка и т. д. превращаются в по-настоящему независимые модули, которые функционируют как отдельные процессы и взаимодействуют с ядром и друг с другом на общих основаниях. Это означает, что имевшее место раньше четкое разделение программного обеспечения на системные и прикладные программы размывается, т. к., фактически, между процессами, реализующими функции ОС, и прикладными процессами, выполняющими программы пользователя, нет никаких различий. Все компоненты системы используют средства микроядра для обмена сообщениями, но взаимодействуют непосредственно. Микроядро лишь проверяет законность сообщений, пересылает их между компонентами и обеспечивает доступ к аппаратуре.

Задания

- Скачать образ дистрибутива minix с официального сайта www.minix3.org.
- Скачать и установить систему виртуализации VirtualBox с сайта www.virtualbox.org. Возможно использование других систем (kvm, wmware, hyperV, VirtualPC).
- В системе виртуализации создать виртуальную машину для minix.
- Установить minix
- Настроить и проверить соединение с Интернет
- Проверить подключение от windows к minix с помощью программ putty и winscp. В случае использования ОС отличной от windows можно использовать другие аналогичные программы (ssh, sftp, nautilus)
- Установить в minix компилятор языка СИ clang, утилиту сборки make.

Подробная инструкция по установке в методических указаниях для лабораторных по Операционным системам.

Содержание отчета

- Скриншот настроек виртуальной машины, описание настроек.
- Скриншот теста Интернет-соединения.
- Скриншот тестовой компиляции с помощью clang.

ЛАБОРАТОРНАЯ РАБОТА 2

Основное содержание работы

В лабораторной работе необходимо обучиться основным навыкам работы в интерактивной оболочке UNIX.

Краткие теоретические сведения

sh - вызывает командный процессор. Shell может работать в двух режимах: интерактивном режиме (режиме командной строки) и режиме выполнения программы. Символ “$” в начале строки обозначает, что shell находится в интерактивном режиме и готов к вводу команд.

Синтаксис:

sh [опции] [программа_на_shell]

Пример:

$ sh prog.sh

cp - копирует файлы. Можно скопировать один файл в другой или список файлов в каталог.

Синтаксис:

cp исходный_файл файл_назначения или

cp исходный_список каталог_назначения

где: исходный_файл - это файл, который нужно скопировать;

файл_назначения - имя файла, в который будет скопирован исходный_файл;

исходный_список - это список файлов для копирования, разделенных пробелами;

каталог_назначения - это каталог, куда копируются файлы.

Пример:

$ cp a.txt /tmp

mv - пересылает или переименовывает файл.

Синтаксис:

mv [-fi] oldfile newfile

mv [-fi] file... destination_directory

Команда mv в первой форме пересылает(или изменяет) файл oldfile с превращением его в newfile.Команда mv во второй форме пересылает один или несколько файлов в каталог destination_directory.
Пример:
$ mv f1.txt f2.txt

who - показывает пользователей, вошедших в систему.
Синтаксис:
who [am i]
Команда who показывает Unix-имена пользователей, которые к данному времени вошли в систему.
Команда who am i показывает ваше входное имя.

echo - берет аргументы, которые ей передали, и записывает их в стандартный вывод.
Синтаксис:
echo [-n] строка
где: -n - это ключ, который подавляет особенность выводить все выходные данные с новой строки.
Пример:
$ echo “Hello All!”

date – устанавливает/выводит системную дату и время.
Синтаксис:
date MDhmY
где: M - месяц(01-12); D - день(01-31); h - час(00-23); m - минуты(00-59); Y - год(00-99).

cat - связывает или соединяет файлы вместе. Эту команду также можно использовать для вывода файла на экран. Если никакие файлы не указаны, команда cat читает со стандартного ввода. Синтаксис:
cat [ключи] [список_файлов]
Пример:
$ cat f1 f2 > f3

mkdir - создает каталог.
Синтаксис:
mkdir название_каталога
Пример:
$ mkdir v10/a

file - определяет тип файла.
Синтаксис:
file [ключи] [список-файлов]
ключи: -f - проверяет файлы из списка

type - указывает физическое нахождение вызванной порграммы.
Пример:

$ type myprog

Результатом действия является: /bin/myprog

find - просматривает список каталогов, отыскивая файлы удовлетворяющие некоторому критерию.
Сиснтаксис:

find [список_каталогов] [спецификация_сопоставления]

Спецификация_сопоставления определяет условия соответствия для искомых файлов и может принимать следующие значения:

-name файл - заставляет команду find искать указанный файл;

-print - выводит имена найденных файлов.

Пример:

$ find . -name myfile -print

(поиск файла с именем myfile ведется в текущем каталоге и его подкаталогах, когда файл найден, на экран выводится полное имя маршрута к нему).

grep - отыскивает текст в файлах.
Синтаксис:

grep [-il][-e]expression[file...]

ключи: -e expression -здесь expression есть отыскиваемый текст. Для задания одного выражения указывать -e перед ним не нужно.Если выражение expression содержит пробелы, они должны быть заключены в кавычки.

-i - игнорирует различие прописных и строчных букв.

-l - вместо показа соответствующих строк выводит только файлы, имеющие соответствия.

Пример: для отыскания всех файлов в текущем каталоге, содержащих слово "john", применяется команда:

$ grep -i john

wc - выводит число строк, слов и символов в файле.
Синтаксис:

wc [-clw] file...

Команда wc подсчитывает число символов, строк и слов в файле или файлах.
Ключи:

-c -подсчитывает только число символов.

-l -подсчитывает только число строк.

-w -подсчитывает только число слов.

cal - отображает календарь в стандартный вывод. По умолчанию берется текущий месяц и год.
Синтаксис:

cal [месяц] [год]

Пример:

$ cal 2000

(выводит календарь на 2000 год).

diff - сравнивает два текстовых файла и сообщает, что должно быть сделано, чтобы один из них стал подобен другому.

Синтаксис:

diff [ключи] старый_файл новый_файл

Задание I

Войти в систему под именем stud (пароль входа stud).

Используя соответствующие команды, получить следующую информацию:

-имя каталога, в котором находитесь в данный момент входа;

-кто работает в системе;

-номера ваших запущенных процессов;

-текущее время и дату;

-типы файлов в текущем каталоге;

-процент занятости дисковой памяти;

-местоположение вызванных ранее команд.

Задание II

Создать на произвольный месяц любого года и записать его в файл f1.

Например, на март 2000 года:

```
cal 3 2000 > f1
```

Создать календарь на год и месяц вашего рождения, результат записать в файл f2.

Объединить полученные файлы и результат записать в файл f3.

Это можно сделать одним из двух способов:

```
cat f1 f2 > f3
```

или

```
cp f1 f3
cat f2 >> f3
```

Записать содержимое каталога /home в файл f4:

```
ls /home > f4
```

Подсчитать (не вручную :-) число строк, слов, символов в файле f2, а результат записать в файл f5. Необходимую для этого команду записать самостоятельно.

Задание III

Используя команду echo, разобрать правила переключения клавиатуры на ввод прописных, строчных, латинских и русских букв.

Задание IV

Выполнить следующие команды с использованием заранее подготовленных файлов.

просмотр файла f1;
сортировка файла f1;
сравнение файлов f1 f3;

Задание V

Ознакомиться с командой bc. Выполнить с ее помощью простейшие математические действия.

Задание VI

По номеру своего варианта выбрать ту структуру каталогов и файлов, которую вам необходимо построить.

С помощью соответствующих команд построить заданную систему каталогов. Поместить полученные в ходе работы файлы (f1, f2, f3, f4, f5) в соответствии с вариантом задания. Удостовериться, что созданная вами структура совпадает с с заданием. Для этого перейти в ваш домашний каталог и выполнить команду ls -R

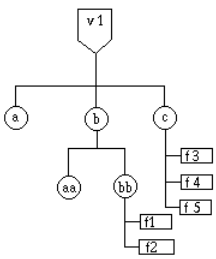
Содержание отчета

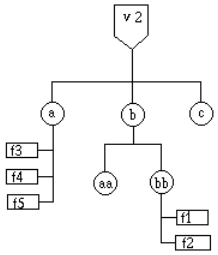
1. Титульный лист: название дисциплины, наименование работы, номер варианта, ФИО студента, дата выполнения.
2. Результаты выполнение задания I: формулировка подзадания, команда для его выполнения, полученный результат.
3. Рисунок схемы, согласно варианту, и последовательность команд в задания VI.

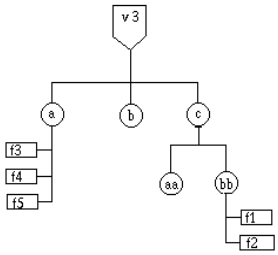
Контрольные вопросы

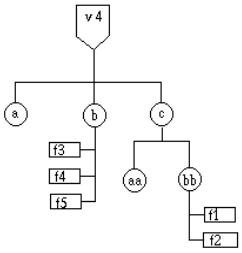
По каким признакам можно определить, что система готова к вводу команд?
Что является командой в shell?
Назовите несколько команда, ориентированных для работы с файловой системой.
Что называют стандартным вводом и выводом?
Как и чем можно переопределить стандартные ввод и вывод?
Что произойдет при вызове: “cat <<cat>>cat”?
Как обозначаются вышестоящий и текущий каталоги?
Как получить быструю справку о команде (не спрашивая преподавателя)?
Как можно использовать bc в неинтерактивном режиме?
Для чего можно использовать результат выполнения команды diff?
Сколько файлов можно сцепить с помощью команды cat?

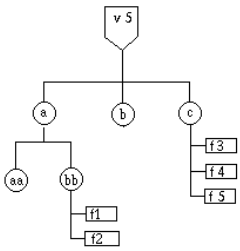
Варианты

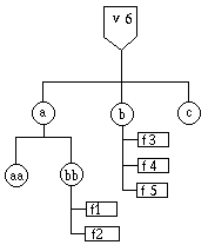

v 1
a
b
c
aa
bb
f 3
f 4
f 5
f1
f2

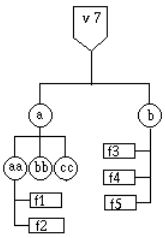

v 2
a
b
c
f3
f4
f5
aa
bb
f1
f2

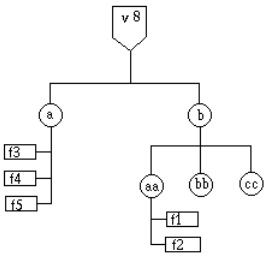

v 3
a
b
c
f3
f4
f5
aa
bb
f1
f2

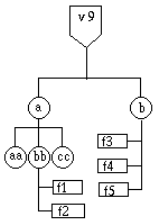

v 4
a
b
c
f3
f4
f5
aa
bb
f1
f2

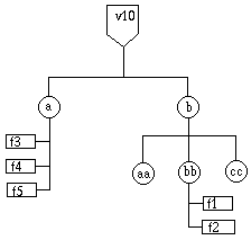

v 5
a
b
c
f3
f4
f 5
aa
bb
f1
f2

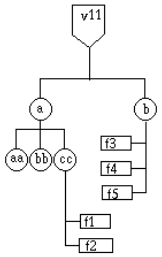

v 6
a
b
c
f3
f4
f 5
aa
bb
f1
f2

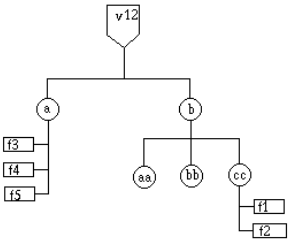

v 7
a
b
aa
bb
cc
f3
f4
f5
f1
f2

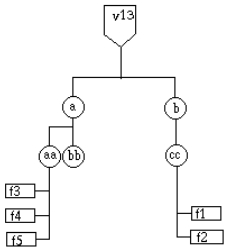

v 8
a
b
f3
f4
f5
aa
bb
cc
f1
f2

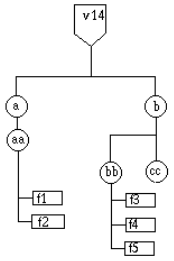

v 9
a
b
aa
bb
cc
f3
f4
f5
f1
f2

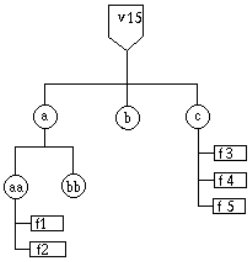

v10
a
b
f3
f4
f5
aa
bb
cc
f1
f2

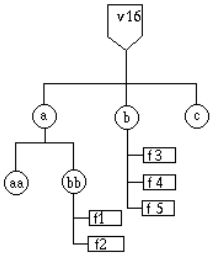

v11
a
b
aa
bb
cc
f3
f4
f5
f1
f2

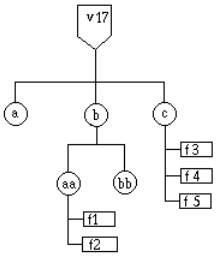

v12
a
b
f3
f4
f5
aa
bb
cc
f1
f2

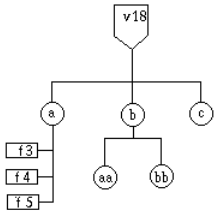

v13
a
b
aa
bb
cc
f3
f4
f5
f1
f2

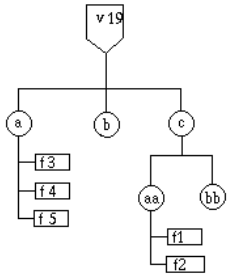

v14
a
b
aa
bb
cc
f1
f2
f3
f4
f5

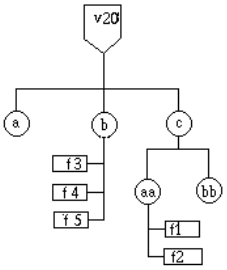

v15
a
b
c
f3
f4
f 5
aa
bb
f1
f2

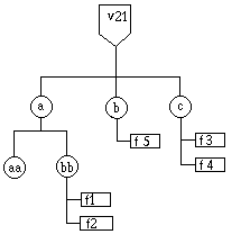

v16
a
b
c
f3
f4
f 5
aa
bb
f1
f2

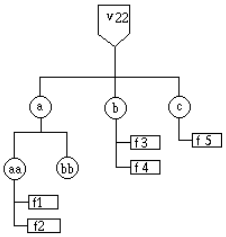

v17
a
b
c
f3
f4
f 5
aa
bb
f1
f2

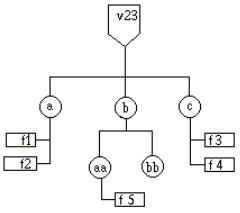

v18
a
b
c
f3
f4
f 5
aa
bb

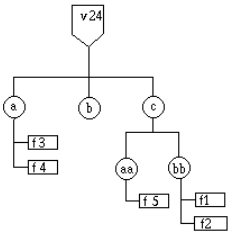

v 19
a
b
c
f3
f4
f 5
aa
bb
f1
f2


v20
a
b
c
f3
f4
f 5
aa
bb
f1
f2


v21
a
b
c
f 5
f3
f4
aa
bb
f1
f2


v22
a
b
c
f3
f 5
f4
aa
bb
f1
f2


v23
a
b
c
f1
f2
f3
f4
aa
bb
f 5


v24
a
b
c
f3
f4
aa
bb
f 5
f1
f2

ЛАБОРАТОРНАЯ РАБОТА 3

Основное содержание работы

Контрольная работа выполняется на макроязыке shell. Желательно (но не обязательно) отладить ее на любой UNIX(Linux)-машине. В работе требуется создавать файловые структуры, научиться перенаправлять ввод/вывод стандартных команд shell.

Краткие теоретические сведения

Простейшие принципы построения программ на языке Shell.

Shell-переменные

Пользователям предоставляется возможность определять переменные и присваивать им значения. Имя переменной должно начинаться с буквы или символа подчеркивания.

Примеры: k7, _ufile, chaos,...

Присвоение значений переменным

Значение переменным присваивается по команде присваивания. Например:

```
$ var=length
```

означает, что переменной var присвоено значение length. Самой length можно в свою очередь, присвоить значение

```
$ length=80
```

Если присваивается значение строки символов, то предпочтительно, чтобы эта строка была заключена в кавычки. Если в строке символов встречаются пробелы, то кавычки обязательны.

В отличие от других языков программирования переменные Shell не связаны с определенным типом данных. Любое значение, которое присваивается переменным Shell, воспринимается как строка символов.

Вывод содержимого переменных

Чтобы получить доступ к переменной, нужно непосредственно перед именем переменной поставить знак доллара($):

```
$ var=80
$ echo $var
80
$
```

Двойные кавычки

Любой символ, который имеет специальное значение для Shell(как, например, *, ?, <, >, >>), утрачивает свой интерпретационный статус. Исключениями здесь являются символ $, обратный апостроф и обратная наклонная черта, если она предшествует специальному символу.

Пример: представим, что текущий каталог содержит следущие файлы:
$ ls
textsdocums sourse
$
Теперь посмотрим результат двух команд echo:
1) $ echo *
textsdocums sourse
$

2) $ echo "*"
*
$
В первом случае Shell заменяет звездочку на имена всех файлов в каталоге. Во втором случае кавычки отменяют специальные значения символа *.

Апостроф
Рассмотрим следующий пример:
$ message="Hello Victor!"
$ echo $message
Hello Victor!
$ echo "$message"
Hello Victor!
$
В обоих случаях получаем одинаковый результат, однако если мы заключим переменную
message в апострофы, то получим следущее:
$ echo '$message'
$message
$
Как видим, когда переменная находится в апострофах, ее значение не подставляется в качестве
результата.

Обратнонаклонная черта и символы(\n, \c, \t)
Обратнонаклонная черта, будучи поставлена перед символом, ликвидирует его специальное значение для Shell.
$ echo \$message
$message
$ echo \\
\
$
Последний пример показывает, что \ можно использовать перед другой \ , чтобы ликвидировать значение последней.
\n - перемещение в следующий ряд;
\c - остаться в той же строке;

\t - перемещение до позиции следующей табуляции;
Пример:
$ echo "1 \t2 \t3 \t4 \t5"
1 2 3 4 5
$

Задание I

Необходимо написать последовательность действий, которая создает файловую структуру первой схемы. Файлы первой схемы должны содержать:

f1 - любимое четверостишье;

f2 - системное время;

f3 - идентификационные имена пользователей, работающих в данный момент с ситемой, и свои имя, фамилию;

f4 - десять символов, являющихся коментарием в языке Shell;

f5 - содержимое всех вышеперечисленных файлов девять раз.

Задание II

Преобразовать полученную файловую структуру в соответствии со второй схемой.

Варианты

Схемы необходимо взять из лабораторной работ в соответствии с нижеследующим распределением на варианты:

Вариант	Первая Схема	Вторая схема	Вариант	Первая Схема	Вторая Схема
1	1	20	15	15	10
2	2	21	16	16	12
3	3	22	17	17	13
4	4	23	18	18	14
5	5	24	19	19	15
6	6	1	20	20	16
7	7	2	21	21	17
8	8	3	22	22	18
9	9	4	23	23	19
10	10	5	24	24	20
11	11	6	25	1	21
12	12	7	26	2	22
13	13	8	27	3	23
14	14	9	28	4	24

Пример выполнения работы для первого варианта

mkdir v1
mkdir v1/a
mkdir v1/b

```
mkdir v1/c
mkdir v1/b/aa
mkdir v1/b/bb
cat <<EOT> v1/b/bb/f1
На голой ветке
Ворон сидит одиноко.
Осенний вечер.
EOT
date > v1/b/bb/f2
who > v1/c/f3
echo “Вася Пупкин” >> v1/c/f3
echo ‘##########’ > v1/c/f4
i=1
while [ $i –lt 10 ]
do
cat v1/c/f[3-4] v1/b/bb/f? >> v1/c/f5
i=`expr $i + 1`
done
mkdir v20
mkdir v20/a
mkdir v20/b
mkdir v20/c
mkdir v20/c/aa
mkdir v20/c/bb
cp v1/b/bb/f* v20/c/aa
cp v1/c/f* v20/b
```

Содержание отчета

1. Титульный лист: название дисциплины, наименование работы, номер варианта, ФИО студента, дата выполнения.
2. Изображение схем и последовательность команд для их преобразования.

Контрольные вопросы

Почему shell называют макроязыком?
Какие еще оболочки ОС вы знаете?
Какие имена переменных допустимы в shell?
Назовите типы переменных shell?
Как получить значение переменной?
В каких случаях используются двойные, а в каких одиночные кавычки?
Для чего используются шаблоны имен файлов?

Лабораторная работа 4

Основное содержание работы

Необходимо научиться создавать сценарии выполнения команд ОС (скрипты) на языке командного интерпретатора shell. Освоить приемы работы с управляющими структурами и переменными shell.

Краткие теоретические сведения

Последовательность команд можно записать в файл и выполнить, указав имя этого файла. Таким образом, обеспечивается возможность создания достаточно сложных программ на языке shell. Чтобы файл успешно запустился, нужно разрешить его выполнение, выставив соотвтствующие права, например командой chmod a+x файл.

В языке существуют операторы управления (оператор цикла, условного перехода) и аппарат передачи параметров.

Оператор for
Общий вид оператора for:
for переменная in список
do
команда
команда
...
done

Значение цикла: выполнение списка команд, заключенного между ключевыми словами "do" и "done", один раз для каждого значения из списка. При этом очередное выбираемое значение из списка присваивается переменной, имя которой указано слева от ключевого слова "in".

Операторы while и until
Оператор while повторяет заданную группу команд, если условие выполнения соответствует истине. Поскольку условие проверяется перед выполнением списка команд, возможна ситуация, когда список не будет выполнен ни разу.
Формат оператора while:
while условие
do
команда
команда
...
done

В операторе until повторение цикла происходит до тех пор, пока проверка условия дает значение false (ложь).

Оператор if

Общий вид оператора if:

```
if условие
then
команда
команда
...
fi
```

Если условие истинно (true), команды, которые находятся между then и fi, будут исполнены,
в противном случае - будут пропущены.

Полный вариант оператора if - else

```
if условие
then
команда
команда
...
else
команда
команда
...
fi
```

Если результат условия true, то исполняются команды, находящиеся между then и else, в противном случае
исполняются команды между else и fi. В обоих случаях используется только одна группа команд.

Команда if может использоваться совместно с командой test.

Оператор case

Оператор case удобен для организации ветвления программы на основе совпадения шаблонов (образцов).

Формат оператора case:

```
case переменная
in
  шаблон1) команда
    команда
    ...
    команда;;
  шаблон2) команда
    команда
```

```
    ...
    команда;;
  шаблонN) команда
    команда
    ...
    команда;;
esac
```

Обратите внимание, что управляющие структуры так же являются командами и у них так же есть входной и выходной поток.
Структура вида имя=значение присваивает значение переменной, например USER=nata, HOME=/usr/nata, Перед исполнением команды, структуры вида $имя заменяются на значение переменной имя.
Для передачи параметров используются некоторые предопределенные переменные окружения:
$n
- n-ный параметр командной строки. $0 – имя самой программы;
$@ - командная строка целиком;
$#
- число параметров командной строки;
$?
- код завершения последней команды;
$$
- номер текущего процесса;

Задание
Разработать скрипт выполняющий задание в соответствии с вариантом.
Скрипт оформить в виде исполняемого файла и отладить на самостоятельно разработанных тестовых примерах.
Содержание отчета
1. Титульный лист: название дисциплины, наименование работы, номер варианта, ФИО студента, дата выполнения.
2. Формулировка индивидуального задания.
3. Исходный текст скрипта
4. Распечатка результатов обработки тестового примера.

Контрольные вопросы

1. Для чего нужна конструкция “#!” в первой строке файла скрипта?
2. Какие реализации интерпретатора shell вы знаете?
3. Как проверить, существует ли заданный файл?
4. Как просмотреть список всех переменных окружения, объявленных в системе?
5. Что произойдет при вызове for i in * do; echo $i; done”?
6. Что произойдет при вызове: “a="echo test"; echo "$a"; echo '$a';

echo `$a`”?
7. Как получить идентификатор текущего процесса в скрипте shell?
8. Как получить имя текущего файла (файла скрипта) в скрипте shell?
9. Как получить все параметры командной строки с которыми был запущен скрипт shell?
10.Можно ли перенаправить ввод или вывод управляющей структуры (например for, if или while)? Что при этом произойдет?

Какие команды можно использовать в качестве условий в if, while, until?

Как будет работать for, если ему не задать список значений?

Можно ли задать значение по умолчанию для case?

Варианты

1. Скрипт выбирает строки, содержащие указанный шаблон, из указанного файла в другой указанный файл. Все данные, необходимые для работы скрипта передаются через аргументы командной строки. Первый аргумент - имя первого файла, второй аргумент - имя второго файла, последующие аргументы - шаблоны поиска. Количество получившихся строк вывести в стандартный поток.
2. Скрипт дописывает в конец трех файлов день недели, год и месяц соответственно. Получившиеся файлы следует сохранить в виде копий, с именами PID.1, PID.2, PID.3,где PID - номер текущего процесса.
3. В подкаталоге "pager" находятся файлы, содержащие различные сообщения для пользователей. Имя каждого файла соответствует имени пользователя, которому предназначено сообщение. Написать скрипт, который проверяет, для всех файлов, работает ли в данный момент нужный пользователь, и, если работает, отправляет ему сообщение из файла. После чего файл удаляется. Предусмотреть возможность отправки сообщения нескольким пользователям одновременно. В этом случае имя файла должно содержать шаблон, определяющий список пользователей. Например: "stud*".
4. Разработать скрипт, исправляющий "ошибки" в тексте. Скрипт принимает во входной поток исходный текст и выдает в выходной поток исправленный текст. В качестве первого аргумента командной строки передается имя файла, содержащего шаблоны исправлений, по 9два регулярных выражения в каждой строке что заменить и на что заменить. Учесть, что части строки, удовлетворяющей первому выражению могут использоваться при замене.Например: ":N" заменить на "<N>>". Где N - любое число. См. man sed и man grep
5. Разработать скрипт-шпион. Скрипт должен запускать программу, заданную первым аргументом командной строки, передавая ей все последующие аргументы. Данные из входного потока скрипта должны передаваться во входной поток программы, а данные с выходного потока программы соответственно в выходной поток скрипта. Кроме того, все передаваемые данные (потоки, включая поток ошибок и

аргументы) должны быть сдублированы в файл <имя_программы>.log
Для удобства чтения секции файла должны быть снабжены комментариями, так же в файле должно быть указано время запуска. Данные о каждом запуске должны дописываться в конец файла.
6. Подсчитать количество строчек в файле /etc/passwd, для которых шеллом прописан /usr/local/bin/bash, записать в файл это число. Подсчитать количество пользователей, для которых шеллом прописан /sbin/nologin, записать в другой файл. Вывести оба эти числа, и их результат их сравнения (больше, меньше, равно).
7. Подсчитать размер своего домашнего каталога и отправить администратору уведомление с этой информацией. Включить в сообщение дату и время. Если размер каталога больше 100 килобайт, удалить в нем все файлы с расширением .o См. man du.
8. Подсчитать количество запущенных httpd (экземпляров веб-сервера), 20 раз в течение 5 минут (через пятнадцать секунд), результаты сохранить в файл, с указанием времени.
9. Вывести на экран список всех графических файлов в текущем каталоге. Не стоит искать такие файлы по расширениям, так как расширения ничего не значат. Обращать внимание нужно на тип содержимого (MIME type). Тип содержимого графических файлов обычно содержит слово "image". См. man file.
10. Разработать скрипт, обеспечивающий интерактивное взаимодействие с пользователем. Скрипт должен выводить приглашение, ожидать команды пользователя, отрабатывать команду, выдавать на экран результаты работы, затем снова выдавать приглашение и ожидать следующей команды. Реализовать обработку следующих команд: проверку существования заданного файла, создание файла, удаление файла, завершение работы со скриптом. Реализовать так же обработку возможных ошибок с выдачей соответствующих сообщений.
11. Василий Пупкин скачал новый аниме-сериал. К сожалению, Василий плохо знает японский, поэтому он скачал русские субтитры. Его проигрыватель не может сопоставить субтитры с видеорядом, так как файлы имеют разные имена. Разработать скрипт, переименовывающий файлы в текущем каталоге, заданные шаблоном. Имена видеофайлов имеют следующий формат:
"<фиксированная_часть_1> - <номер>.avi"
Имена субтитров имеют следующий формат:
"<фиксированная_часть_2> - <номер>.ssa"
Где: <фиксированная_часть> - любой набор символов,
<номер> - две любые цифры (обычно от 01 до 26)
Нужно переименовать все субтитры, чтобы имена файлов совпадали с именами видеофайлов, отличаясь только расширениями. Все параметры передавать в аргументах командной строки. Учесть, что кроме указанных выше в каталоге у Василия могут находиться и другие файлы.

12. В текущем каталоге среди различных файлов находятся файлы, хранящие пароли пользователей для доступа к различным службам. Эти файлы имеют различные имена, похожа лишь их структура. Каждая строка такого файла содержит пару: <имя_пользователя>:<хэш_пароля> Найти все такие файлы. Список файлов вывести на экран.
13. Просмотреть все .c файлы в текущем каталоге, скопировать в каталог test все файлы, содержашие слово "marked"
14. Перемешать строки файла, заданного первым аргументом командной строки в случайном порядке.Результат вывести в стандартный поток.
15. Осуществить резервное копирование файлов, указанных в файле victims.dat в каталог ~/yyyy-mm-dd, где yyyy - текущий год. mm - текущий месяц. dd - текущий день. Копировать только те файлы, которые "новее" (дата модификации больше), чем victims.dat См. man stat, man find.
16. Файл, заданный первым аргументом командной строки содержит список сотрудников. Каждая строка файла содержит фамилию, имя и отчество одного сотрудника, разделенные пробелами. Требуется напечатать приглашения на корпоративную вечеринку для каждого сотрудника. Приглашения вывести в файл, заданный вторым аргументом командной строки для последующей распечатки.
17. Разработать скрипт, подготавливающий файлы для резервного копирования. В первом аргументе командной строки скрипт получает размер носителя, и выводит на экран список файлов из текущего каталога, начиная с самых старых, чтобы их суммарный размер соответствовал заданному. При повторном запуске скрипт выбирает следующую порцию файлов. То есть, нужно запоминать последний выведенный файл или дату его модификации. См. man stat, man find, man touch.
18. Вывести в файл, заданный аргументом, список пользователей, работающих в данный момент с системой. Строки пронумеровать.
19. Для каждого пользователя, зарегистрированного в системе проверить, работает ли он в данный момент. Всем работающим послать сообщение "Hello.", список "прогульщиков" отослать на e-mail администратору. Учесть, что не все пользователи, указанные в /etc/passwd, являются людьми. Отличительные признаки людей - /bin/bash в качестве шелла и домашний каталог, являющийся подкаталогом /home.
20.Слить два файла, заданных аргументами командной строки, так чтобы в результирующем третьем файле находилась первая строка первого файла, затем первая строка второго файла, затем вторая строка первого файла, затем вторая строка второго файла и так далее.

Лабораторная работа 5

Основное содержание работы

Научиться использовать утилиту AWK для обработки табличной информации.

Краткие теоретические сведения

Язык AWK используется для комбинированной обработки символьных и числовых полей в записях. В результате генерируется отчет в запланированной программистом форме. Программы на языке AWK можно эффективно использовать как фильтры данных для преобразования вывода одной программы и передачи результата фильтрации на вход другой.
Команда имеет формат:
awk [-Fсимвол] [[-f] программа] [аргумент ...] [файл ...]
Команда awk сопоставляет строки исходных файлов с правилами, определенными в программе. Правила можно задать либо непосредственно в командной строке, либо поместить в файл с именем программа и воспользоваться опцией -f. Если шаблоны указаны в командной строке, их следует заключить в одинарные кавычки ('), чтобы избежать интерпретации shell'ом.
Каждое правило имеет вид:
шаблон { действие }
Каждая исходная строка сопоставляется с каждым из шаблонов; в случае успеха выполняются указанные действия. После сопоставления со всеми шаблонами вводится следующая строка и процесс сопоставления повторяется. Может быть опущен либо шаблон, либо действие, но не оба вместе. Если для данного шаблона не указаны действия, то строка просто копируется на стандартный вывод. Если для действия не определен шаблон, 12то оно будет выполняться для каждой входной строки. Строки, которые не удалось сопоставить ни одному шаблону, игнорируются.
Шаблон - это произвольная логическая комбинация, составленная с помощью операций !, ||, && и скобок из регулярных выражений и выражений сравнения. Регулярные выражения обрамляются символами /
Действие есть последовательность операторов. Так как шаблоны и действия могут быть опущены, то, чтобы различать их в программе, последние надо брать в фигурные скобки.
Оператор есть одна из конструкций:
if (условие) оператор [else оператор]
while (условие) оператор
for (выражение; условие; выражение) оператор
break
continue
{ [оператор] ... }
переменная = выражение
print [список_выражений] [> выражение]

printf формат [, список_выражений] [> выражение]
next # пропустить оставшиеся шаблоны и перейти к
следующей строке
exit # пропустить оставшиеся строки
Более полные руководства по использованию AWK можно найти здесь:
http://moshkow.perm.ru/win/MAN/DEMOS210/awk.txt
http://citforum.perm.ru/operating_systems/manpages/AWK.1.shtml
http://citforum.perm.ru/operating_systems/articles/sed_awk.shtml
Задание
Разработать скрипт, разбирающий лог (журнал) http сервера (файл:
/var/log/apache/stud_access_log). Каждая строка журнала состоит из
следующих полей, разделенных пробелами:
– адрес, с которого поступил запрос;
– имя пользователя, если пользователь авторизовался, или
прочерк;
– прочерк;
– дата в квадртных скобках, состоит из:
– собственно, даты (например 05/Mar/2004:12:13:12);
– часового пояса (например +0500).
– строка запроса в кавычках, состоит из:
– ключевого слова (например GET – запрос файла);
– параметров (в случае с GET – имя запрашиваемого
файла);
– версии протокола (например HTTP/1.1);
13– кода результата обработки запроса (например 200 – успешно);
– количества переданных клиенту байт.
Получить и вывести на экран результаты обработки запроса в
соответствии с вариантом. При разработке скрипта, shell использовать только для обработки параметров вызова и запуска AWK. Обработку информации осуществлять средствами AWK. Все необходимые параметры передавать в аргументах командной строки. Скрипт оформить в виде исполняемого файла и отладить на самостоятельно разработанных тестовых примерах.

Содержание отчета

1. Титульный лист: название дисциплины, наименование работы,
номер варианта, ФИО студента, дата выполнения.
2. Формулировка индивидуального задания.
3. Исходный текст скрипта
4. Распечатка результатов обработки тестового примера.

Варианты

1. Вывести количество успешных обращений к указанному ресурсу.
2. Вывести количество хитов (запросов) за указанный период.
3. Вывести количество хостов (уникальных адресов, с которых были
запросы), за указанный период.

4. Вывести количество визитов (уникальных адресов в день), за указанный период.
5. Вывести имя ресурса, к которому было больше всего неудачных обращений.
6. Вывести коэффициент загрузки канала, которым сервер подключен к Интернет (отношение объема реально переданных данных к максимально возможному объему), считая, что пропускная способность канала 10 Мбит/с.
7. Вывести количество удачных обращений к указанному ресурсу за указанный период.
8. Найти 10 самых больших по объему запрошенных ресурсов.
9. Вывести адрес, с которого было произведено больше всего удачных запросов.
10. Определить, по каким числам, четным или нечетным, пользуется большей популярностью заанный ресурс.
11. Вывести уникальные имена всех ресурсов, запрошенных из заданной подсети.
12. Вывести сумму номеров строк, в которых встречается заданный код ошибки.
13. Вывести данные об ошибках обработки запросов в следующем формате: порядковый номер, код ошибки, сколько раз происходила ошибка, когда последний раз происходила ошибка.
14. Вывести суммарный объем данных, переданных сервером за указанный период.
15. Вывести суммарный объем данных, переданных сервером на указанный адрес.
16.Вывести имя пользователя, который проработал с системой дольше всех за указанный период.
17. Вывести среднее количество обращений в месяц к указанному ресурсу.
18. Вывести максимальное количество обращений в месяц с указанного адреса.
19. Вывести код самой распространеной ошибки.
20. Вывести адрес, с котрого было сделано больше всего ошибочных запросов.

Лабораторная работа 6

Основное содержание работы

Научиться использовать простейшие средства разработки и поддержки проекта (gcc, make, gdb). Научиться использовать механизм сокетов для обмена информацией между приложениями.

Краткие теоретические сведения

Утилита make позволяет поддерживать, изменять и регенерировать группы программ. Make выполняет команды из make-файла, который можно задать, используя опцию -f make_файл Если отсутствует опция -f, то ищутся файлы makefile, Makefile, и файлы системы управления исходными текстами (SCCS) s.makefile и s.Makefile в указанном порядке. В командной строке может встретиться более чем одна пара -f make_файл. Основным "строительным элементом" make-файла являются правила (rules). В общем виде правило выглядит так:
<цель_1> <цель_2> ... <цель_n>: <зависимость_1>
<зависимость_2> ... <зависимость_n>
<команда_1>
<команда_2>
...
<команда_n>
Цель (target) - это желаемый результат, способ достижения которого описан в правиле. Цель может представлять собой имя файла. В этом случае правило описывает, каким образом можно получить новую версию этого файла.
Зависимость (dependency)- это "исходные данные", необходимые для достижения указанной в правиле цели. Можно сказать что зависимость - это 15"предварительное условие" для достижения цели. Зависимость может представлять собой имя файла. Этот файл должен существовать, для того чтобы можно было достичь указанной цели. Команда_1, команда_2, ..., команда_n – это команды shell, которые должны быть выполнены для достижения цели. Строки, содержащие команды должны начинаться с двух символов табуляции.
Чтобы скомпилировать программу, написанную на языке «Си» и получить исполняемый файл можно использовать компилятор cc. Команда cc принимает аргументы следующих типов:
Аргументы, оканчивающиеся на .c, интерпретируются как имена файлов, содержащих исходные тексты на языке «Си»; они компилируются, и каждый об ектный модуль помещается в файл с именем, которое образуется из имени исходного файла замещением расширения .c на .o. Созданная выполняемая программа, если не указана опция редактора связей -o, помещается в файл a.out.
Командой cc интерпретируются следующие опции:
-c – отменить фазу редактирования связей и создавать объектный файл даже в случае программы, состоящей только из одного модуля.
-g – сгенерировать дополнительную информацию для отладчика.
-o файл – сохранить созданную программу в файл.
-O – включить оптимизацию об ектного кода.
-L каталог – дополнить каталогом список каталогов, которые содержат об ектные библиотечные модули.

-l библиотека – скомпоновать с об ектной библиотекой.
-I каталог – дополнить каталогом список каталогов, которые содержат включаемые (посредством директивы #include) файлы.
Примеры:
cc test.c – компилирует исходный текст, содержащийся в файле test.c. Скомпилированная прогрмма помещается в исполняемый файл a.out.
cc test.c -c -o xxx.o – компилирует исходный текст, содержащийся в файле test.c. Результатом будет объектный файл xxx.o.
Для отладки программы можно использовать отладчик gdb.
Программные гнезда (sockets) - Поддерживаемый ядром механизм, скрывающий особенности сетевой среды и позволяющий единообразно взаимодействовать процессам.
Создание сокета:
sd=socket(domain, type, protocol);
domain - домен гнезда
type - тип (с виртуальным соединением или дейтаграммное)
protocol - желаемый сетевой протокол
Функция возвращает дескриптор созданного сокета или ошибку.
Закрытие (уничтожение) гнезда
close(sd)
Связывание ранее созданного программного гнезда с именем:
bind(sd, socknm, socknlen);
sd - дескриптор ранее созданного программного гнезда
socknm - адрес структуры, которая содержит имя (идентификатор) гнезда, соответствующее требованиям домена данного гнезда и используемого протокола
для домена системы UNIX имя является именем объекта в файловой системе при создании программного гнезда создается файл
socknlen - длина в байтах структуры socknm
Запрос связи с существующим программным гнездом со стороны процесса-клиента:
connect(sd, socknm, socknlen);
смысл параметров, как у функции bind
имя программного гнезда на другой стороне коммуникационного канала
у гнезда с дескриптором sd и у гнезда с именем socknm должны быть одинаковые домен и протокол
если тип гнезда с дескриптором sd - дейтаграммный, то connect служит для информирования системы об адресе назначения пакетов, которые в дальнейшем будут посылаться с помощью функции send
Информирования о том, что процесс-сервер планирует установление виртуальных соединений через указанное гнездо:
listen(sd, qlength);
qlength - максимальная длина очереди запросов на установление соединения, которые должны буферизоваться системой, пока их не выберет

процесс-сервер
Выборка процессом-сервером очередного запроса на установление соединения с указанным программным гнездом:
nsd = accept(sd, address, addrlen);
sd - дескриптор существующего программного гнезда, для которого ранее была выполнена функция listen
address - массив данных, в который должна быть помещена информация, характеризующая имя программного гнезда клиента
addrlen - адрес, по которому находится длина массива address
выполнение функции приводит к установлению виртуального соединения
nsd - новый дескриптор программного гнезда, который должен использоваться при работе через данное соединение
по адресу addrlen помещается реальный размер массива данных, которые записаны по адресу address
Передача и прием данных через программные гнезда с установленным виртуальным соединением:
count = send(sd, msg, length, flags);
count = recv(sd, buf, length, flags);
В send:
msg указывает на буфер с данными, которые требуется послать
length - длина этого буфера
flags – параметры передачи
В recv:
buf указывает на буфер, в который следует поместить принимаемые данные
length - максимальная длина этого буфера
flags – параметры приема
Немедленная ликвидация установленного соединения:
shutdown(sd, mode);
немедленно остановить коммуникации в зависимости от значения параметра mode
Более подробные руководства по использованию механизма сокетов, make, gcc и gdb можно найти здесь:
http://moshkow.perm.ru/win/MAN/DEMOS210/make.txt
http://citforum.perm.ru/operating_systems/manpages/MAKE.1.shtml
http://citforum.perm.ru/operating_systems/gnumake/index.shtml
http://citforum.perm.ru/operating_systems/manpages/CC.1.shtml
http://citforum.perm.ru/operating_systems/manpages/SDB.1.shtml
http://citforum.perm.ru/programming/c_unix/index.shtml
http://citforum.perm.ru/operating_systems/bach/contents.shtml

Задание

Разработать приложение, осуществляющее обмен информацией при помощи механизма сокетов на стороне клиента или сервера в соответствии с

заданием. Необходимо, чтобы ваш проект состоял из нескольких модулей, поэтому процедуру, выполняющую основные действия, вынесите в отдельный файл. Для сборки проекта необходимо создать Makefile и использовать утилиту make. Для отладки программы используйте утилиту debug. В качестве клиента для отладки программы-сервера используйте telnet, в качестве сервера для отладки клиента - chargen (например здесь: 195.19.162.93:19).

Контрольные вопросы

1. В чем отличие блокирующих и неблокирующих сокетов?
2. Какие методы обработки на стороне сервера нескольких соединений одновременно вы знаете?
3. Для чего используются абстрактные (PHONY) цели в Makefile?
4. Какая переменная содержит имя цели обрабатываемого правила при обработке Makefile?
5. Какой диапазон TCP портов доступен пользователю для создания соединения на стороне сервера?
6. Как выглядит адрес UNIX сокета?
7. В чем отличие передачи датаграмм от устновления виртуального соединения?
8. Назовите полное имя файла библиотеки, которая подключется компилятором при указании ключа -lm.

Содержание отчета

1. Титульный лист: название дисциплины, наименование работы, номер варианта, ФИО студента, дата выполнения.
2. Формулировка индивидуального задания.
3. Исходный текст скрипта
4. Распечатка результатов обработки тестового примера.

Варианты

1. Реализовать службу Discard. На стороне сервера открыть соединение, принимать любые данные "в никуда". Реализовать возможность обработки нескольких соединений.
2. Реализовать службу Chargen. Генератор символов. На стороне сервера открыть соединение, после этого выдавать клиенту все отображаемые ASCII символы по очереди с максимально возможной скоростью до закрытия соединения клиентом.
3. Реализовать службу Echo. На стороне сервера открыть соединение. Возвращать клиенту посланные им данные до закрытия соединения клиентом.
4. Реализовать службу Daytime. На стороне сервера открыть соединение. Выдать клиенту текущую дату и время в виде текстовой строки в формате удобном для понимания человеком.
5. На стороне сервера установить соединение с клиентом, принять от него команду в виде текстовой строки, выполнить, результат

выполнения вернуть клиенту.
6. Написать клиент для службы Daytime (см. вариант 4). Установить соединение с сервером, получить текущую дату и вывести ее на экран.
7. Написать клиент для службы Chargen (см. вариант 2). Установить соединение с сервером, получить 1 Мб данных, закрыть соединение. Вывести на экран время передачи данных для измерения пропускной способности канала.
8. Написать клиент для службы Discard (см. вариант 2). Установить соединение с сервером, передать 1 Мб данных, закрыть соединение. Вывести на экран время передачи данных для измерения пропускной способности канала.
199. Написать клиент для службы из задания 5. Передать на сервер заданную команду, принять и выдать на экран результат ее выполнения.
10. Реализовать чат на два пользователя. На стороне клиента принимать данные пользователя и отправлять их на сервер. На стороне сервера в свою очередь установить соединение с клиентом, принимать данные и выводить их на экран.
11. Реализовать простой "прокси" сервер. На стороне сервера открыть соединение. Принять от клиента запрос вида: "GET ...". Установить соединение с сервером, указанным в запросе , передать ему тот же самый запрос, принять данные и вернуть их клиенту.
12. Реализовать простой HTTP сервер. На стороне сервера открыть соединение. Принять от клиента запрос вида: "GET ...". Если в указан заданный сервер и запрошенный файл существует в текущем каталоге, выдать клиенту этот файл. Перед выдачей файла выдать строку "Content-type: text/plain\n\n"
13. Реализовать простой сервер сбора данных. На строне сервера установить соединение с клиентом, принять строку текста, сохранить в файл с указанием текущего времени. Предусмотреть возможность одновременной обработки нескольких соединений.
14. Реализовать сканер открытых TCP портов. Пытаться установить соединение с заданным сервером с указанием порта из заданного диапазона. Вывести на экран адрес сервера и результаты попыток для каждого порта.
15. Реализовать сканер открытых TCP портов. Пытаться установить соединение с каждым сервером из заданного диапазона с указанием заданного порта. Вывести на экран результаты попыток для каждого сервера.
16. Реализовать службу Finger. Принять имя пользователя. Выдать информацию об этом пользователе.
17. Создать два процесса. Первый процесс запрашивает у пользователя имя файла, второй создает файл с этим именем и записывает в него текущую дату. Обмен информацией между процессами осуществляется при помощи UNIX сокета.

18. Отправить заданный файл через SMTP сервер
19. Получить заголовки писем с POP3 сервера
20. Реализовать сервер для игры в "крестики-нолики" принимать от пользователей ходы, выдавать текущую ситуацию на поле, контролировать выигрыш одного из игроков. Реализовать возможность подключения двух игроков.

Лабораторная работа 7

Создание процессов

Лабораторная работа 8

Взаимодействие процессов

Лабораторная работа 9

Семафоры

Лабораторная работа 10

Грамматики