Пермский филиал федерального государственного автономного образовательного учреждения высшего образования "Национальный исследовательский университет "Высшая школа экономики"

МЕТОДИЧЕСКИЕ УКАЗАНИЯ И КОНТРОЛЬНЫЕ ЗАДАНИЯ

по дисциплине «Информационные процессы системы и сети»

Пермь, 2020

Аннотация

Методические указания содержат лабораторные работы, которые позволяют освоить навыки программирования на языке командного интерпретатора shell, изучить основные навыки работы с инструментами компиляции программ, отлаживать программы с помощью gdb, использовать профилятор gprof, создавать и организовывать взаимодействие процессов**.**

Каждая лабораторная работа состоит из следующих частей:

- основное содержание работы;
- краткие теоретические сведения;
- задание;
- контрольные вопросы;
- варианты.

Работа рекомендована для студентов 2 курса по специальности Бизнес Информатика. А также данный материал будет полезен для преподавателей.

Оглавление

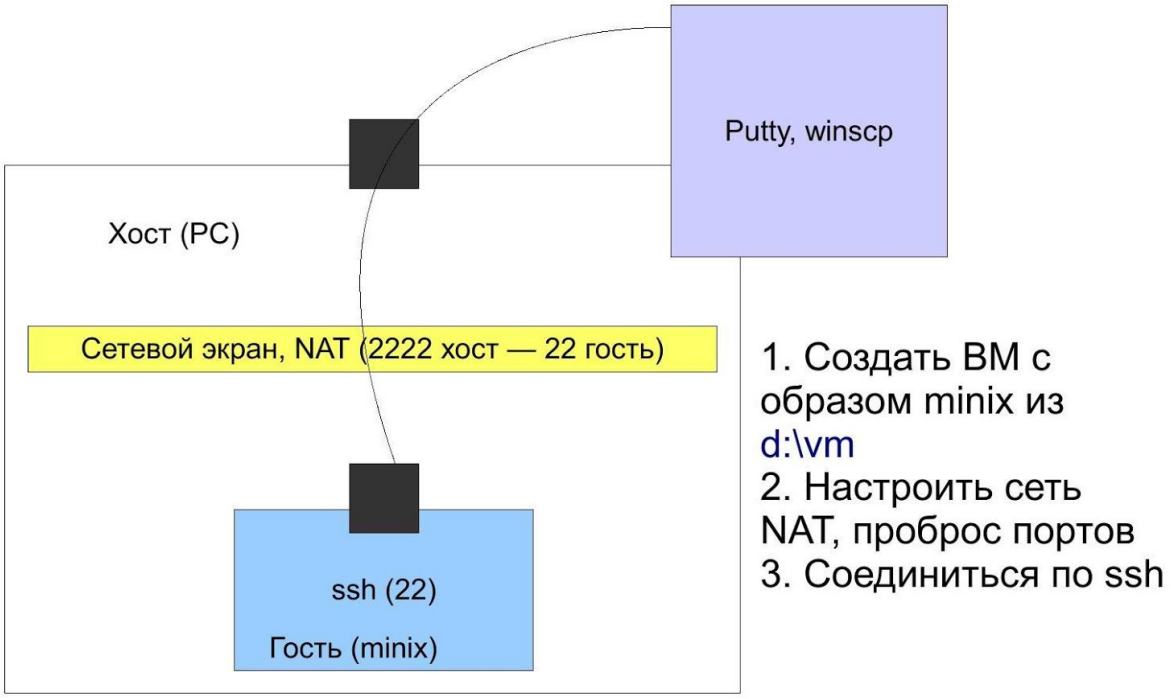
Лабораторная работа №1 виртуальные машины

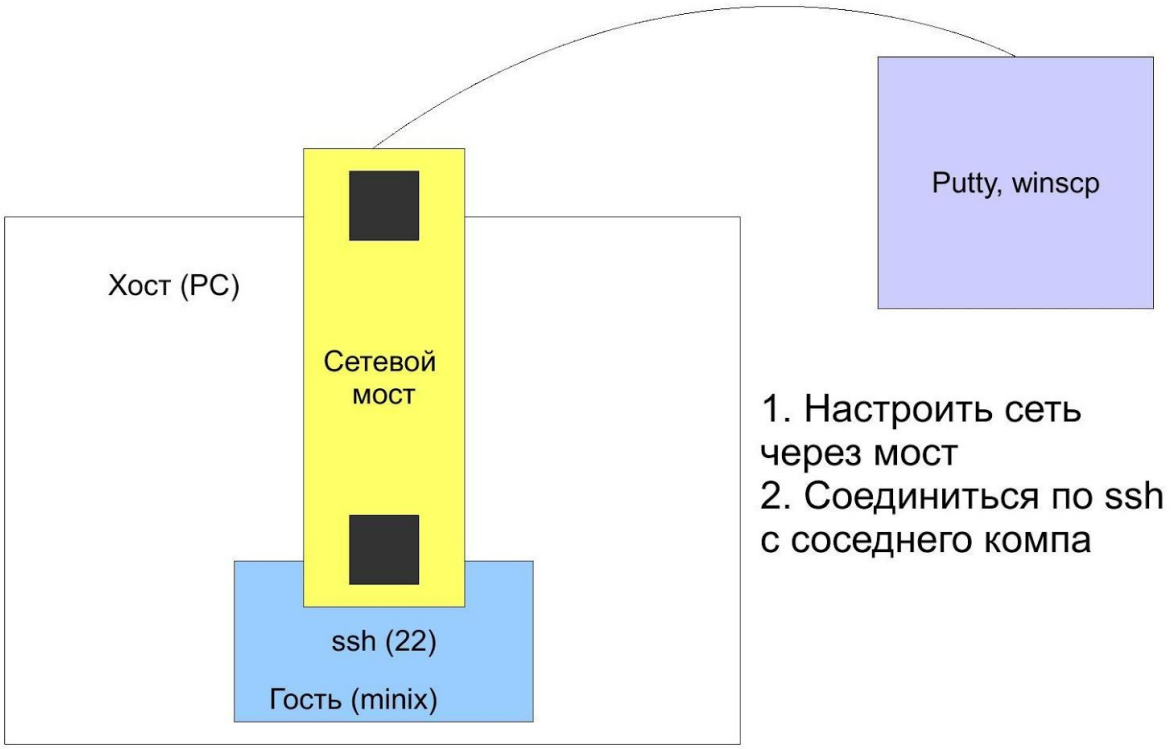
Основное содержание работы

В лабораторной работе необходимо обучиться основным навыкам работы с виртуальными машинами.

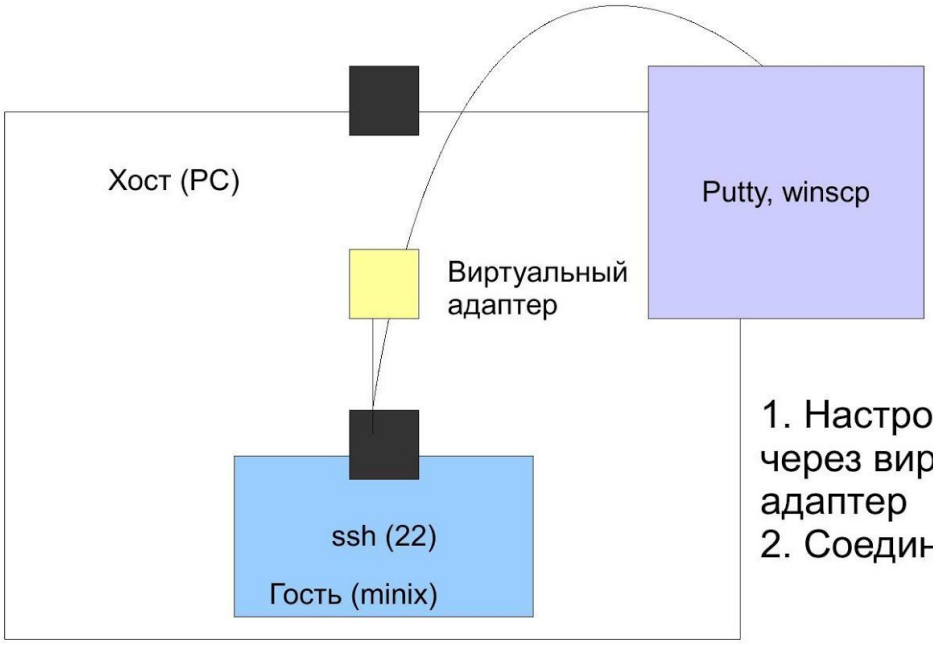
Задание

Настроить виртуальные машины согласно схемам. Желательно последовательно выполнить настройки по каждой из схем. Образ диска для виртуальной машины скачать с kdenisb.org (оставайтесь на раздаче! ;). Используемые программные средства (для windows): virtualbox, putty, winscp

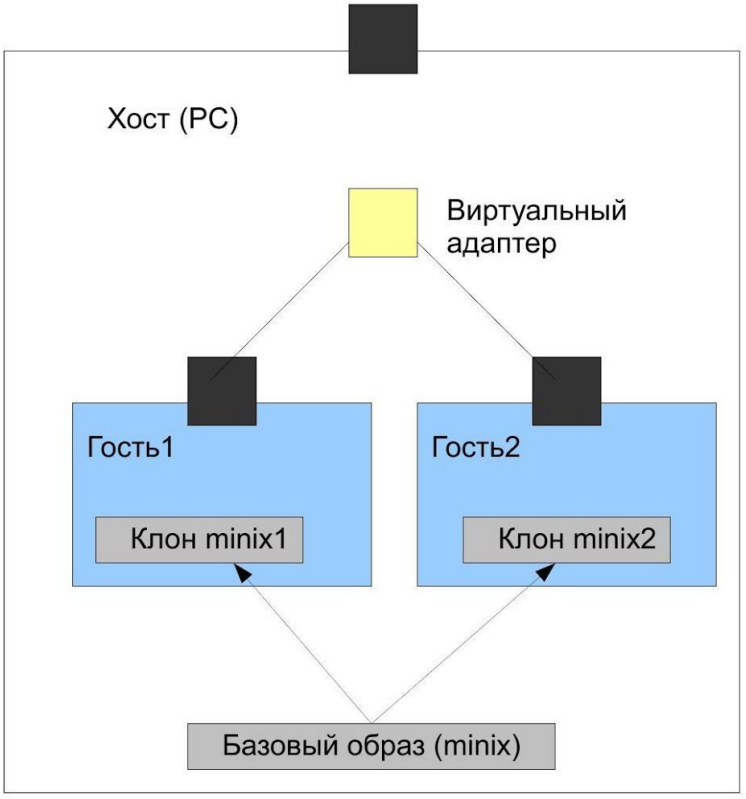




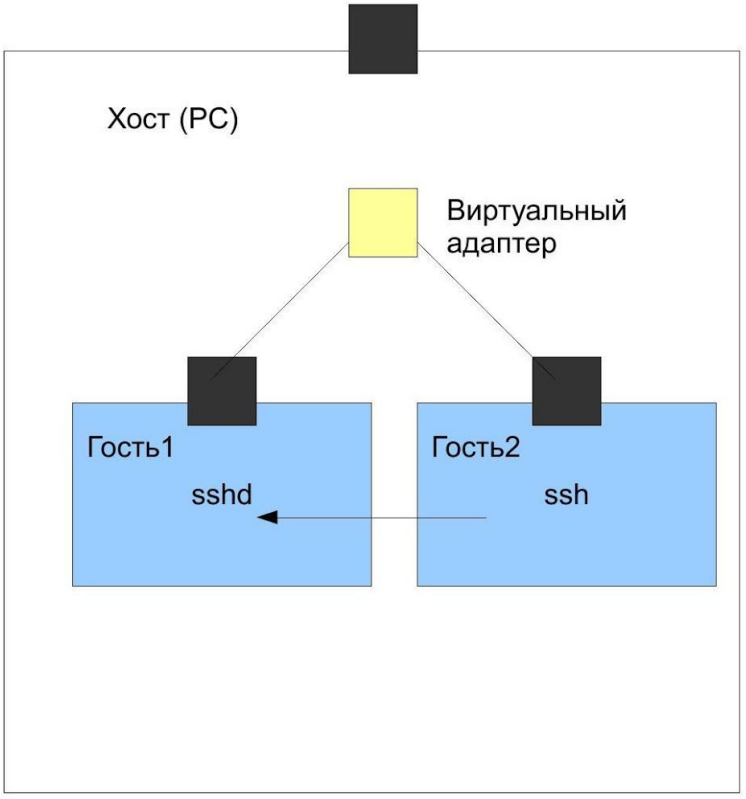
1. Настроить сеть через мост
2. Соединиться по ssh с соседнего компа



1. Настроить сеть через виртуальный адаптер
2. Соединиться по ssh



1. Сделать два клона
От образа диска minix
2. Настроить сеть
3. Проверить ping
4. Убедиться в различности дисков



Организовать соединение по ssh между minix 1 и 2 по публичным ключам.

https://www.debian.org/devel/passwordlessssh

/usr/pkg/etc/ssh/ssh_config
StrictHostKeyChecking no

Лабораторная работа №2 командная оболочка

Основное содержание работы

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

Краткие теоретические сведения

Shell - это стандартный командный язык программирования, который выполняет чтение команд с терминала или из файла. Как только вы подключитесь к системе, вы увидите приглашение командной строки shell. Обычно это значок $. После этого можно вводить команды. Ниже - список некоторых полезных команд shell.

cal - отображает календарь в стандартный вывод.

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.

cp - копирует файлы.

date – устанавливает/выводит системную дату и время.

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

echo - берет аргументы, которые ей передали, и записывает их в стандартный вывод.

file - определяет тип файла.

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

grep - отыскивает текст в файлах.

ls – показывает содержимое каталога

mkdir - создает каталог.

mv - пересылает или переименовывает файл.

pwd – показывает имя текущего каталога.

sh - вызывает командный процессор.

Shelltype - указывает физическое нахождение вызванной программы.

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

who – показывает список пользователей, вошедших в систему.

Получить подробную информацию по использованию этих и других команд можно при помощи команды *man*, передав ей в качестве параметра имя интересующей вас команды. Например: *man man*

Shell так же предоставляет средства для перенаправления входного и выходного потока команд при помощи специальных символов: > - перенаправление вывода, < - перенаправление ввода, | - перенаправление вывода одной команды на ввод другой. Например *ls> f1* сохраняет содержимое текущего каталога в файле f1. *wc -l < f2* подсчитывает 4 количество строк в файле f2. *who | wc -l* подсчитывает количество пользователей.

Задание

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

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.

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

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

Объединить полученные файлы и результат записать в файл f3.

Записать содержимое каталога /home в файл f4:

Подсчитать число строк, слов, символов в файле f2, а результат записать в файл f5.

С использованием ранее подготовленных файлов выполнить следующие команды.

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

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.

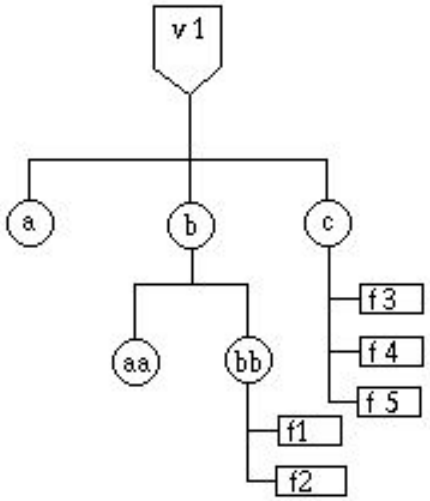
2. Результаты выполнение задания: формулировка подзадания, команда для его выполнения, полученный результат.

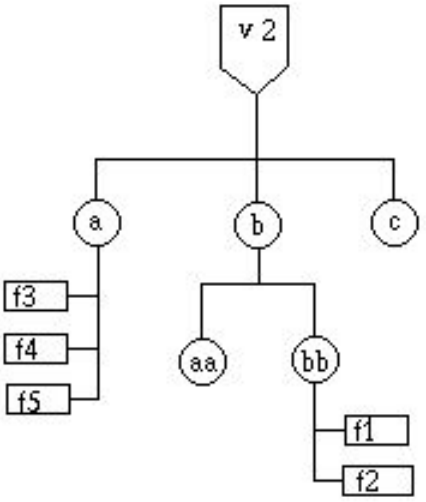
3. Рисунок схемы, согласно варианту, и последовательность команд для создания заданной структуры каталогов.

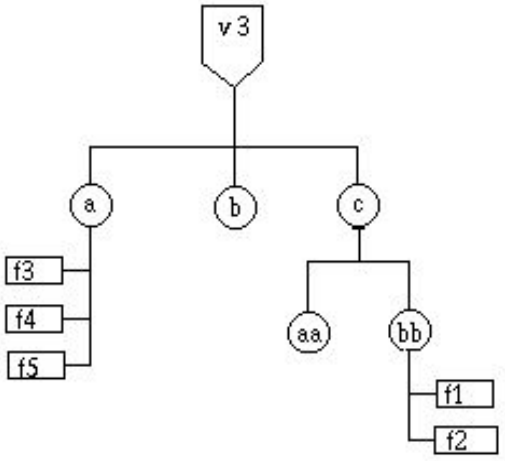
Контрольные вопросы

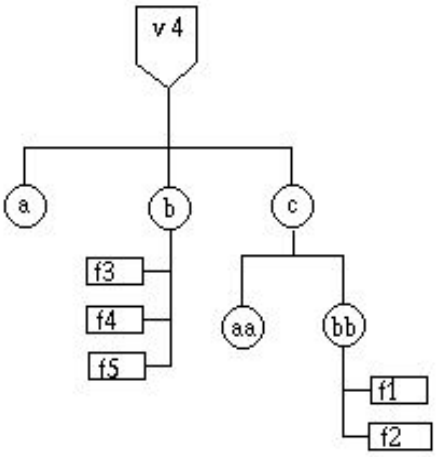
1. По каким признакам можно определить, что система готова к вводу команд?
2. Что является командой в shell?
3. Назовите несколько команда, ориентированных для работы с файловой системой.
4. Что называют стандартным вводом и выводом?
5. Как и чем можно переопределить стандартные ввод и вывод?
6. Что произойдет при вызове: “cat<<cat>>cat”?
7. Как обозначаются вышестоящий и текущий каталоги?
8. Как можно использовать bc в не интерактивном режиме?
9. Для чего можно использовать результат выполнения команды diff?
10. Сколько файлов можно сцепить с помощью команды cat?

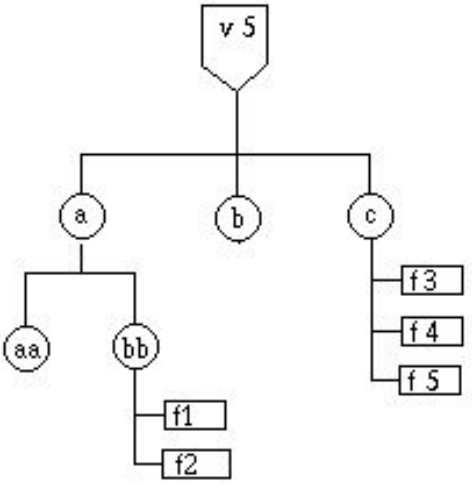
Варианты

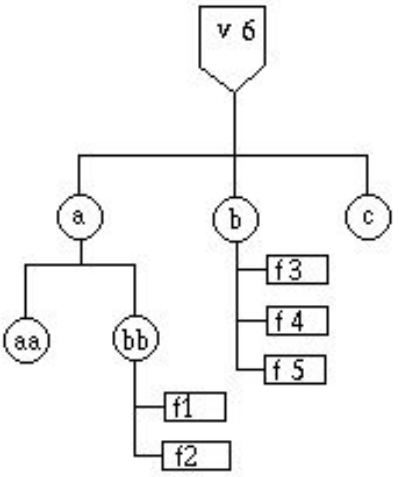

v 1
a
b
c
aa
bb
f 3
f 4
f 5
f1
f2

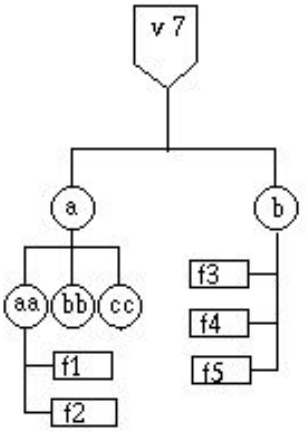

v 2
a
b
c
f3
f4
f5
aa
bb
f1
f2

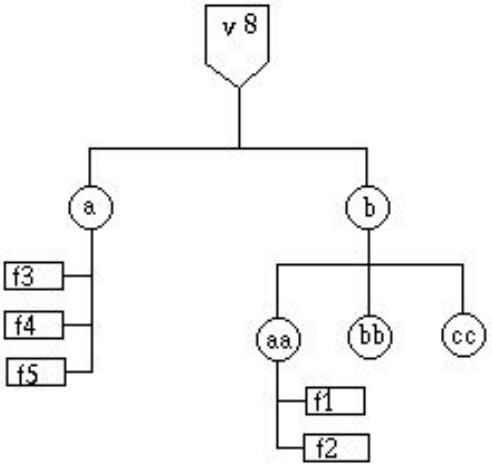

v 3
a
b
c
f3
f4
f5
aa
bb
f1
f2

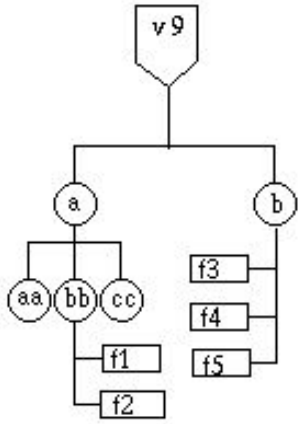

v 4
a
b
c
f3
f4
f5
aa
bb
f1
f2

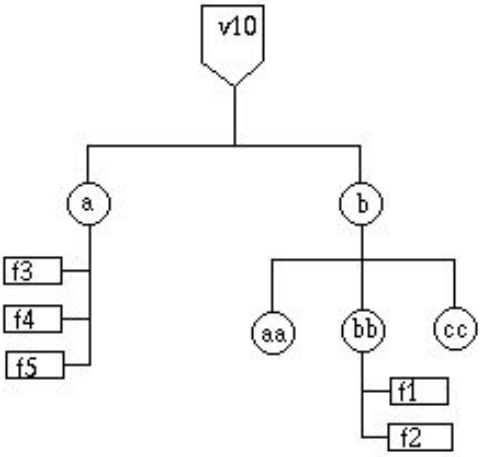

v 5
a
b
c
f 3
f 4
f 5
aa
bb
f1
f2

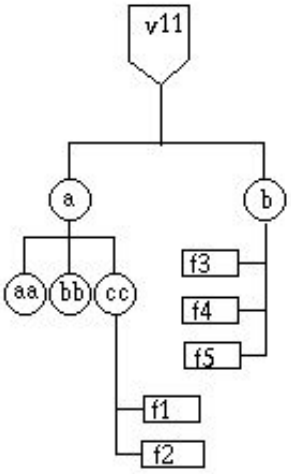

v 6
a
b
c
f 3
f 4
f 5
aa
bb
f1
f2

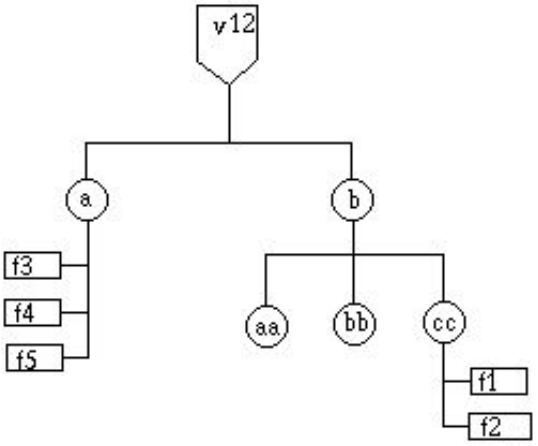

v 7
a
b
aa
bb
cc
f3
f4
f5
f1
f2

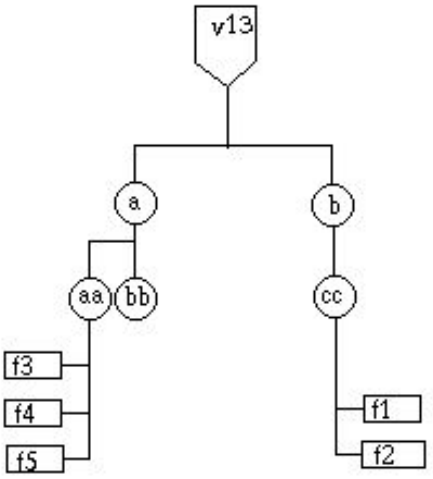

v 8
a
b
f3
f4
f5
aa
bb
cc
f1
f2

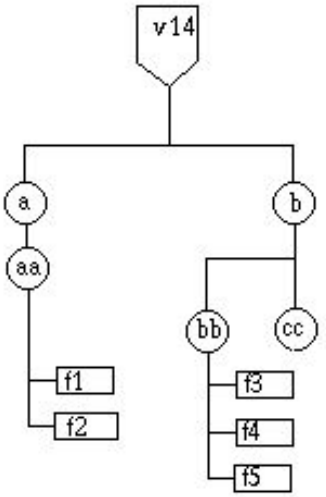

v9
a
b
aa
bb
cc
f1
f2
f3
f4
f5

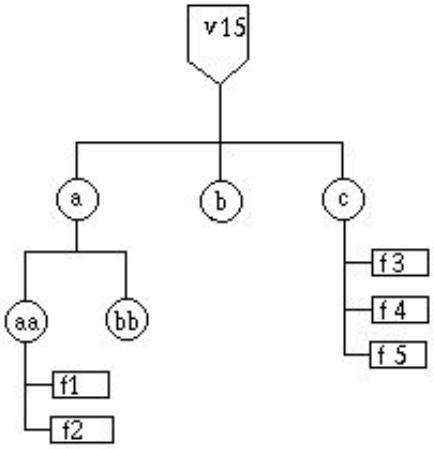

v10
a
b
f3
f4
f5
aa
bb
cc
f1
f2

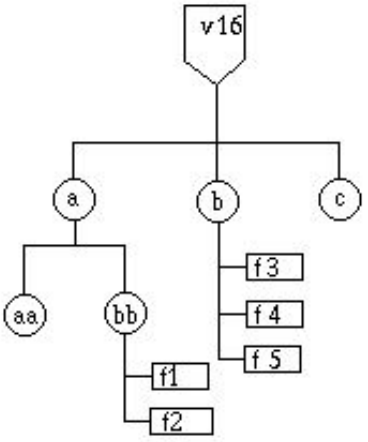

v11
a
b
aa
bb
cc
f3
f4
f5
f1
f2

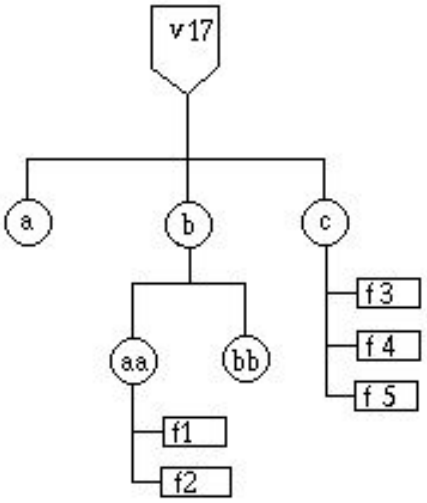

v12
a
b
f3
f4
f5
aa
bb
cc
f1
f2

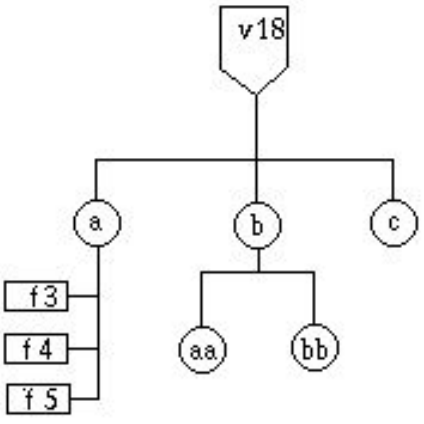

v13
a
b
aa
bb
cc
f3
f4
f5
f1
f2

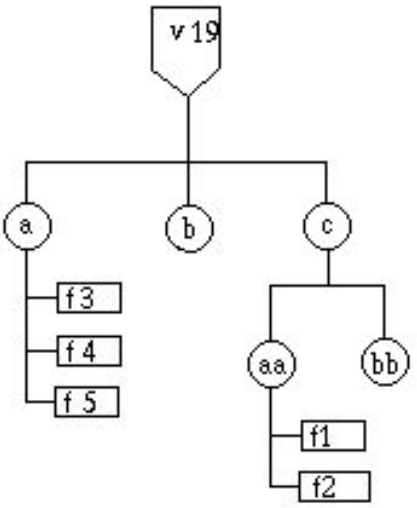

v14
a
b
aa
bb
cc
f1
f2
f3
f4
f5

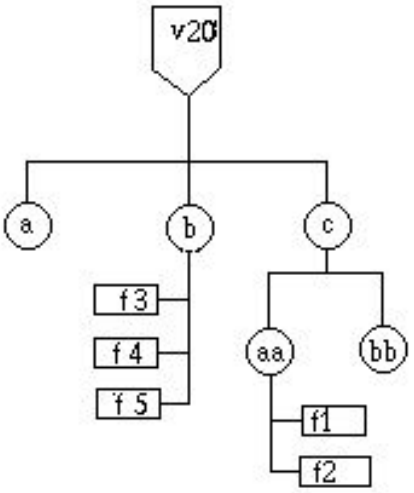

v15
a
b
c
aa
bb
f1
f2
f3
f4
f5


v16
a
b
c
aa
bb
f1
f2
f3
f4
f5


v17
a
b
c
aa
bb
f1
f2
f3
f4
f5


v18
a
b
c
f3
f4
f5
aa
bb


v19
a
b
c
f3
f4
f5
aa
bb
f1
f2


v20
a
b
c
f3
f4
f5
aa
bb
f1
f2

Лабораторная работа №3 разработка сценариев

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 соответствующие права, например командой chmoda +x файл.

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

Операторы цикла:

for имя [in слово1 слово2 ...]
do список_команд
done

Переменная имя последовательно принимает значения из списка слов. Для каждого значения имя выполняется *список_команд*.

while список_команд_1
[do список_команд_2]
done

Выполняет список_команд_2, пока список_команд_1 успешно выполняется.

Оператор условного перехода:

if список_команд_1
then список_команд_2
[else список_команд_3]
fi

Если *список_команд_1* успешно выполняется, то выполняется*список_команд_2* иначе выполняется*список_команд_3*. Команда if может использоваться совместно с командой test.

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 скрипт eshell?
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?

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 два 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. mandu.

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, содержащие 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. manstat, manfind.

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. manstat, manfind, mantouch.

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 №4 текстовый процессор awk

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, то 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

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 квадратных 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);

– 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 заданный 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 распространенной ошибки.

20. Вывести адрес, с которого было сделано больше всего ошибочных запросов.

Лабораторная работа №5 сборка программных систем - Makefile

Цель работы

Изучение основных навыков работы с инструментами компиляции программ, функционала *make.*

Краткие теоретические сведения

Make — утилита, автоматизирующая процесс преобразования файлов из одной формы в другую. Чаще всего это компиляция исходного кода в объектные файлы и последующая компоновка в исполняемые файлы или библиотеки.

Утилита использует специальные make-файлы, в которых указаны зависимости файлов друг от друга и правила для их удовлетворения. На основе информации о времени последнего изменения каждого файла make определяет и запускает необходимые программы.

Программа make выполняет команды согласно правилам, указанным в специальном файле. Этот файл называется make-файл (makefile, мейк файл). Как правило, make-файл описывает, каким образом нужно компилировать и компоновать программу.

make-файл состоит из правил и переменных. Правила имеют следующий синтаксис:

```
цель1 цель2 ...: реквизит1 реквизит2 ...
 команда1
 команда2
 ...
```

Правило представляет собой набор команд, выполнение которых приведёт к сборке файлов-целей из файлов-реквизита.

Правило сообщает make, что файлы, получаемые в результате работы команд (цели) являются зависимыми от соответствующих файлов-реквизита. make никак не проверяет и не использует содержимое файлов-реквизита, однако, указание списка файлов-реквизита требуется только для того, чтобы make убедилась в наличии этих файлов перед началом выполнения команд и для отслеживания зависимостей между файлами.

Обычно цель представляет собой имя файла, который генерируется в результате работы указанных команд. Целью также может служить название некоторого действия, которое будет выполнено в результате выполнения команд (например, цель clean в make-файлах для компиляции программ обычно удаляет все файлы, созданные в процессе компиляции).

Строки, в которых записаны команды, должны начинаться с символа табуляции.

Рассмотрим следующий пример: существует каталог v0, содержащий в себе файл prog.c. В файле prog.c содержится исходный код программы, написанной на C:

```
//включаем header “myv0lib.h” в программу
#include “myv0lib.h”
//основная функция, исполняемая при запуске
int main()
{
    int x,y; //объявили x и y, типа int
    char s[64]; //объявили массив s длиной 64
    //функции, вызываемые далее, хранятся в библиотеке myv0lib, описаны соответствующим header’ом
    x=myv0f1(6); //в переменную x сохраняем результат выполнения функции myv0f1 от значения 6
```

y=myv0f2(7,8); //в переменную y сохраняем результат выполнения функции myv0f2 от значений 7, 8

sprintf(s, "x=%d y=%d\n",x,y); //в переменную s записывается строка приведенного в кавычках формата, символы %d заменяются аргументами x, y

write(1,s,sizeof(s)); //вывод на экран строки

return 0; //завершение программы с результатом 0 (как правило, означающим стандартное закрытие)

}

В папке v0 так же содержится папка *libs*, содержащая в себе следующие файлы:

myv0f1.c – файл на языке C, описывающий функцию myv0f1.c;

myv0f2.c – файл на языке C, описывающий функцию myv0f1.c;

myv0lib.h – header файл, описывающий библиотеку myv0lib;

Необходимо скомпилировать с помощью clang программу, созданную с помощью вышеуказанных файлов. Для этого необходимо создать Makefile для библиотеки myv0lib и программы prog, после чего выполнить их с помощью утилиты make.

Сперва необходимо создать Makefile для компиляции библиотеки. Для этого, сперва нужно перейти в каталог *libs* и выполнить в нем команду *nano Makefile* для того, чтобы создать необходимый файл. Затем, файл необходимо заполнить следующим образом:

```
#Libs Make
all: libmyv0
myv0f1.o: myv0f1.c
	clang –c –omyv0f1.o myv0f1.c
myv0f2.o: myv0f2.c
	clang –c –omyv0f2.o myv0f2.c
libmyv0: myv0f1.o myv0f2.o
	ar r ../libmyv0.a myv0f1.o myv0f2.o
	ranlib ../libmyv0.a
clean:
	rm –f *.o
```

При запуске этого файла утилитой *make* (утилита автоматически подхватывает файлы, названные как «Makefile» в той директории, в которой была запущена) будет выполнена следующая последовательность действий:

1. Сперва будет выполнена первая цель, описанная в файле, а именно – цель *all*.

2. Для достижения цели *all* необходим реквизит *libmyb0*, который в данный момент времени отсутствует. Соответственно, будет запущена цель *libmyv0*.

3. Для достижения цели *libmyv0* необходимы реквизиты *myv0f1.o* и *myv0f2.o*, которые так же отсутствуют. Соответственно, последовательно будут запущены цели *myv0f1.o* и *myv0f2.o*.

4. Для достижения цели *myv0f1.o* необходим реквизит *myv0f1.c*, который есть в папке, а значит цель можно достигнуть. Соответственно, будет выполнена *команда clang c –omyv0f1.o myv0f1.c*, которая, используя компилятор *clang* создаст объект *myv0f1.o*, используя исходный код из файла *myv0f2.c*.

5. Аналогично 4 пункту, будет достигнута цель *myv0f2.o*.

6. После появления реквизитов из 4 и 5 пункта, можно достигнуть цель *libmyv0*. Соответственно, будет выполнена команда *ar r ../libmyv0.a myv0f1.o myv0f2.o*, которая заархивирует файлы *myv0f1.o myv0f2.o* в файл *../libmyv0.a*, а затем командой *ranlib* архив индексируется.

7. После выполнения цели из пункта 6 становится возможным достигнуть цели *all*, поскольку реквизит в наличии. Однако, в цели *all* не указано команд, которые необходимо выполнить, соответственно, скрипт просто завершится.

В ходе выполнения вышеописанной последовательности действий не была затронута цель *clean*. Эту цель можно выполнить командой *make clean*. Сама цель не требует реквизитов, и лишь удалит из каталога все файлы с расширением *.o*, удалив таким образом результаты компиляции файлов *myv0f1.c* и *myv0f2.c*.

Теперь, необходимо скомпилировать саму программу prog, при этом указав что необходимо использовать библиотеку libmyv0. Для этого, сперва необходимо перейти к директорию с файлом prog.c, и создать в этой директории Makefile со следующим содержимым:

```
#main Make
all: prog
prog: prog.c
	clang –oprog prog.c –lmyv0 –Ilibs/ -L.
clean:
	rm –f prog
```

Так, при запуске *make* будет выполнена компиляция и получен файл prog, который можно запустить командой

```
./prog
```

Однако, такой подход к компиляции – сперва библиотека, затем сама программа – не слишком удобен в случае, если необходимо компилировать несколько библиотек. Удобнее было бы сразу указать в *Makefile*, что необходимо компилировать библиотеку. Для этого, файл *Makefile* должен выглядеть следующим образом:

```
#main Make
all: libmyv0.a prog
libmyv0.a:
	cd libs && make
prog: prog.c
	clang –oprog prog.c –lmyv0 –Ilibs/ -L.
clean:
	cd libs && make clean
	rm –f prog libmyv0.a
```

Во-первых, теперь перед тем как выполнять цель *prog* необходимо выполнить цель *libmyb.a*, в ходе которой будет запущен *make*-файл в директории с библиотекой, и уже после этого будет выполнена цель *prog*. Так же, была модифицирована цель *clean* для вызова цели *clean* из директории с библиотекой, и удаления скомпилированной библиотеки. Для того, чтобы проверить работу make- файла, сперва необходимо выполнить команду *make clean* для удаления всего, что было скомпилировано до этого, а затем – команду *make*, чтобы собрать программу с нуля и убедиться, что все работает.

Синтаксис, поддерживаемый утилитой *make*, так же позволяет использовать переменные. Например, чтобы объявить переменную, необходимо написать в *make*-файле до объявления целей:

```
X=
```

Для того, чтобы использовать переменную впоследствии, нужно будет написать *${X}*. Тогда, при выполнении скрипта утилитой, вместо *${X}* будет подставлено значение переменной *X*.

Например, можно модифицировать make-файл в директории с файлом prog.c следующим образом:

```
#main Make
CFLAG=-Ilibs/ -L. //объявили переменную
all: libmyv0.a prog
libmyv0.a:
	cd libs && make
```

prog: prog.c

clang –oprog prog.c –lmyv0 ${CFLAG} //использовали переменную

clean:

cd libs && make clean

rm –f prog libmyv0.a

Более того, переменные можно объявлять в отдельных файлах. Например, создадим файл *Makefile.inc* со следующим содержимым:

CC=clang //объявили переменную

Изменим файл *Makefile* следующим образом:

#main Make

include Makefile.inc //подключили файл в котором происходит объявление переменных

CFLAG=-Ilibs/ -L. //объявили переменную

all: libmyv0.a prog

libmyv0.a:

cd libs && make

prog: prog.c

${CC} –oprog prog.c –lmyv0 ${CFLAG} //использовали переменную CC и CFLAG

clean:

cd libs && make clean

rm –f prog libmyv0.a

Так же изменим make-файл, находящийся в папке *libs* следующим образом:

#Libs Make

include ../Makefile.inc //подключили файл в котором происходит объявление переменных

all: libmyv0

myv0f1.o: myv0f1.c

${CC} –c –omyv0f1.o myv0f1.c //использовали переменную CC

myv0f2.o: myv0f2.c

${CC} –c –omyv0f2.o myv0f2.c //использовали переменную CC

libmyv0: myv0f1.o myv-f2.o

ar r ../libmyv0.a myv0f1.o myv0f2.o

ranlib ../libmyv0.a

clean:

*rm –f *.o*

Задание к работе

Создайте каталог MyProg, в нем – каталог libs. В каталоге MyProg создайте файл prog.c, в каталоге libs – файлы myv0f1.c, myv0f2.c, myv0lib.h. Содержимое файлов должно соответствовать приложению А.

Создайте make-файлы для библиотеки в каталоге libs и программы в каталоге MyProg.

Создайте собственную библиотеку в каталоге mylib, используя язык С, и модифицируйте программу prog таким образом, чтобы она использовала как библиотеку из папки libs, так и вашу библиотеку, выводя на экран результаты выполнения функций.

Создайте make-файл для вашей библиотеки и модифицируйте make файл в каталоге MyProg для автоматической компиляции вашей библиотеки и всего проекта.

В случае, если при компиляции проекта clang выводит warning сообщения – модифицируйте проект таким образом, чтобы этих предупреждений не возникало. Библиотека должна содержать в себе 3 функции:

Первая функция получает значение int, умножает его на M и выводит результат.

Вторая функция получает два значения int, и из большего из них вычитает 1.

Третья функция получает два значения int, и к меньшему из них прибавляет M.

Каждая функция должна быть описана в отдельном файле. Значение M должно быть задано в header-файле, и соответствовать номеру студента в списке группы. Отчет должен содержать исходные коды созданных и модифицированных в рамках выполнения задания файлов (.c, .h, make-файлов), скриншот с результатом компиляции и скриншот с результатом работы программы.

Лабораторная работа №6 создание процессов

Содержание работы

Работа предназначена для исследования способов создания процессов.

Необходимо выполнить следующие задания:

1. Создать процесс способом по варианту.
2. Получить информацию о коде завершения порожденного процесса.

Варианты

Варианты	API	Среда	Тип процесса
1	winapi	Visual studio	нить
2	winapi	mingw	нить
3	posix	cygwin	нить
4	posix	linux	нить
5	posix	minix	нить
6	winapi	Visual studio	тяжелый процесс
7	winapi	mingw	тяжелый процесс
8	posix	cygwin	тяжелый процесс
9	posix	linux	тяжелый процесс
10	posix	minix	тяжелый процесс

Лабораторная работа №7 взаимодействие процессов

Содержание работы

Цель работы: научиться организовывать взаимодействие двух процессов в среде Windows.
Необходимо выполнить следующие задачи:

1. Создать два процесса, тип процесса определить по номеру варианта.
2. Процесс ПЕРВЫЙ должен передать ВТОРОМУ сигнал событие.
3. ВТОРОЙ процесс должен послать первому произвольную строку, используя способ взаимодействия процессов согласно варианту.
4. ПЕРВЫЙ процесс должен напечатать на экране полученные данные.

Варианты

Варианты	Тип процесса	Способ взаимодействия
1	нить	Неименованный (Анонимный) канал
2	процесс	Неименованный (Анонимный) канал
3	нить	Именованный канал
4	процесс	Именованный канал
5	нить	Мейлслот
6	процесс	Мейлслот
7	нить	Маппирование файла
8	процесс	Маппирование файла

П р и л о ж е н и е

CAT

cat [OPTION] [FILE]...

`cat /etc/passwd` // показать содержание файла

`cat test test1` //посмотреть содержание нескольких файлов в терминале

`cat >test2` //создание файла с командой cat

`cat -n song.txt` //отображение номеров строк в файле

`cat -e test` // отображение $ в конце файла (можно увидеть с опцией-e, что "$" появляется в конце линии, а также если есть разрыв между пунктами. Эта опция полезна для того, чтобы собрать несколько строк в одну строку.)

`cat test >> test1` // Добавление в существующий файл символами ' >> '(двойные больше). Здесь содержимое в файле test будет добавлено в конце файла test1.

`cat test test1 test2 > test3` // Перенаправление содержимого нескольких файлов в один файл

`cat test test1 test2 test3 | sort > test4` // Сортировка содержимого нескольких файлов в один файл

LS

`ls` //без выбора списка файлов и каталогов в голом формате, в котором мы не будем иметь возможности просматривать такие детали, как типы файлов, размер, изменение даты и времени, разрешения и ссылки и т.д.

ls –l // показывает файл или каталог, размер, дату изменения и время, имя файла или папки и владельца файла и его разрешение.

ls –a // Список всех файлов, включая скрытые файлы, которые начинаются с '.'.

ls –lh // показывает размеры в удобном для человека формате.

ls –lS // отображает размер файла по порядку, начиная с наибольшего.

ECHO

echo Linux Open Source Software Technologies // вывод строки на экран

echo -e "Linux \bopen \bsource \bsoftware \btechnologies" //с помощью опции -e можно включить интерпретацию специальных последовательностей. Последовательность \b позволяет удалить предыдущий символ. Например, удалим все пробелы из строки

echo -e "Linux \topen \tsource \tsoftware \ttechnologies" //С помощью \t вы можете добавить горизонтальные табуляции

echo -e "Linux \tnopen \tnsource \tnsoftware \tntechnologies" //Можно совместить переводы строки и табуляции

echo -e "Linux \ropen source software technologies" //С помощью последовательности \r можно удалить все символы до начала строки

Вы можете разукрасить вывод *echo* с помощью последовательностей управления цветом Bash. Для доступны такие цвета текста:

\033[30m - чёрный;
\033[31m - красный;
\033[32m - зелёный;
\033[33m - желтый;
\033[34m - синий;
\033[35m - фиолетовый;

\033[36m - голубой;
\033[37m - серый.

И такие цвета фона:

\033[40m - чёрный;
\033[41m - красный;
\033[42m - зелёный;
\033[43m - желтый;
\033[44m - синий;
\033[45m - фиолетовый;
\033[46m - голубой;
\033[47m - серый;
\033[0m - сбросить все до значений по умолчанию.

Например. раскрасим нашу надпись в разные цвета:

```
echo -e "\033[35mLinux \033[34mopen \033[32msource \033[33msoftware \033[31mtechnologies\033[0m"
```

FIND

```
find ./test   //поиск в указанной папке
find . -name "*.jpg"  //Искать файлы по имени в текущей папке
find . -iname "test*"  //Не учитывать регистр при поиске по имени
find ./test ./test2 -type f -name "*.c   //Искать в двух каталогах одновременно
find ~ -type f -name ".*"  //поиск скрытых файлов
```

TEST

```
if test -f test.txt
then
        rm test.txt
else
        echo 'файл test.txt не найден'  /*пример удалит обычный файл test.txt если он существует, либо выведет сообщение что данного файла нет.*/
```

-b file — истина, если file существует и является специальным блочным устройством.

-c file — истина, если file существует и символьное устройство.

-d file — истина, если file существует и является каталогом.

-e file — истина, если file существует.

-f file — истина, если file существует и является обычным файлом.

-g file — истина, если file существует и имеет установленным групповой идентификатор (set-group-id).

-k file — истина, если file имеет установленным «sticky» бит.

-L file — истина, если file существует и является символьной ссылкой.

-p file — истина, если file существует и является именованным каналом (pipe).

-r file — истина, если file существует и читаем.

-s file — истина, если file существует и имеет размер больше, чем ноль.

-S file — истина, если file существует и является сокетом.

-t [fd] — истина, если fd открыт на терминале. Если fd пропущен, по умолчанию 1 (стандартное устройство вывода).

-u file — истина, если file существует и имеет установленным бит пользователя (set-user-id).

-w file — истина, если file существует и записываем.

-x file — истина, если file существует и исполняем.

-O file — истина, если file существует и его владелец имеет эффективный идентификатор пользователя.

-G file — истина, если file существует и его владелец имеет эффективный идентификатор группы.

file1 -nt file2 — истина, если file1 новее (дата модификации), чем file2.

file1 -ot file2 — истина, если file1 старее, чем file2.

file1 -ef file2 — истина, если file1 и file2 имеют то же устройство и номер inode .

-z string — истина, если длина string равна нулю.

-n string — истина, если длина string не ноль.

string1 = string2 — истина, если строки равны.

string1 != string2 — истина, если строки не равны.

! expr — истина, если выражение expr ложь.

expr1 -a expr2 — истина, если оба выражения expr1 и expr2 истина.

expr1 -o expr2 — истина, если хотя бы одно из выражений expr1 или expr2 истина.

arg1 %оператор% arg2 -eq, -ne, -lt, -le, -gt, или -ge — эти арифметические бинарные операции возвращают истину, если arg1 равно, не равно, меньше чем, меньше чем или равно, больше чем, или больше или равно, чем arg2, соответственно. arg1 и arg2 могут быть положительными целыми, отрицательными целыми, или специальными выражениями -l string, которые вычисляют длину string.

LESS

less [+номер_строки] [+/образец_поиска] [файл...]

h – вызов справки

q – выход

SPACE – на экран вперёд

b – пролистать экран назад

k и j или стрелки вверх и вниз – вертикальная прокрутка по строке

g – перейти на 1-ую строку

<N>g – перейти на строку N

G – перейти на последнюю строку

F – перейти на последнюю строку файла и ожидать записи новых строк (аналог tail -f, не работает со стандартным вводом)

/pattern – поиск по шаблону вперёд

?pattern – поиск по шаблону назад

n – следующее совпадение

N – предыдущее совпадение

mбуква – отметить позицию буквой

'буква (апостоф и буква) – перейти на отмеченную позицию

! – запуск shell-команды (% – имя текущего файла, к примеру: ! cat % > /tmp/foobar.txt)

MORE

more [+номер_строки] [+/образец_поиска] [файл...]

h – help

q – выход

SPACE – пролистать один экран

d – пролистать пол экрана

RETURN – пролистать одну строку

/pattern – пролистать до строки содержащей подстроку pattern

n – повторить поиск подстроки

= – показать номер текущей строки
:n – перейти к следующему файлу
:p – вернуться к предыдущему файлу
:f – напечатать имя текущего файла и номер строки в нем
! – запуск shell-команды

GREP

grep '12:00' /home/david/backup/log.txt //поиск по заданному критерию
-v - Выводи реверсивные результаты. Вместо того чтобы вывести строки где искомое было найдено — выводи те строки где искомой подстроки нет.
-c - Отключает стандартный способ вывода результата и вместо этого отображает только число обозначающее количество найденых строк.
-i - Делает поиск регистронезависимым
-w - Ведет поиск по цельным словам. Например при обычном поиске строки ‘wood’ grep может найти слово ‘hollywood’. А если используется данный ключ то будут найдены только строки где есть слово ‘wood’.
-l - Выводит только имена файлов где была найдена строка.
-r - Производит поиск рекурсивно по всем поддиректориям.

CHMOD

Число		е обозначения
0	разрешения отсутствуют	---
1	x — запуск	--x
2	w — изменение	-w-
3	x+w — запуск+изменение	-wx
4	r — чтение	r--
5	r+x — чтение+запуск	r-x
6	r+w — чтение+изменение	rw-
7	r+w+x — чтение+изменение+запуск	rwx

Запись прав доступа символами
Примеры: *chmod +x myfile1*
chmod g=rw myfile2
chmod u-w myfile3

Обозначения для владельцев файла следующие:
u - Владелец-пользователь.
g - Группа.
o - Все остальные.
a - Вообще все.

Математические операции означают следующее:
+ - Добавляет к текущим правам доступа новое разрешение.
- - Удаляет из текущих прав доступа определенное разрешение.
= - Устанавливает полностью новые разрешения (предыдущие перезаписываются новыми).

CHOWN

chown новый_владелец имя_файла //Для изменения владельца файла
chown новый_владелец:новая_группа имя_файла //Для изменения владельца и группы файла

MKDIR

mkdir [-m режим_доступа] [-p] каталог //создание каталогов

Командой mkdir обрабатываются две опции:
-m режим_доступа - Явное задание режима_доступа для создаваемых каталогов [см. chmod].
-p - При указании этой опции перед созданием нового каталога предварительно создаются все несуществующие вышележащие каталоги.

RMDIR

rmdir [-p] [-s] каталог ... //удаление каталогов

Командой rmdir обрабатываются следующие опции:
-p - Позволяет удалить каталог и вышележащие каталоги, оказавшиеся пустыми. На стандартный вывод выдается сообщение об удалении всех указанных в маршруте каталогов или о сохранении части из них по каким-либо причинам.
-s - Подавление сообщения, выдаваемого при действии опции -p.

RM

rm [-f] [-i] файл ... //удаление файлов
rm -r [-f] [-i] каталог ... [файл ...]

Допускаются следующие три опции:
-f - Команда не выдает сообщений, когда удаляемый файл не существует, не запрашивает подтверждения при удалении файлов, на запись в которые нет прав. Если нет права и на запись в каталог, файлы не удаляются. Сообщение об ошибке выдается лишь при попытке удалить каталог, на запись в который нет прав (см. опцию -r).
-r - Происходит рекурсивное удаление всех каталогов и подкаталогов, перечисленных в списке аргументов. Сначала каталоги опустошаются, затем удаляются. Подтверждение при удалении файлов, на запись в которые нет прав, не запрашивается, если задана опция -f или стандартный ввод не назначен на терминал и не задана опция -i. При удалении непустых каталогов команда rm -r предпочтительнее команды rmdir, так как последняя способна удалить только пустой каталог. Но команда rm -r может доставить немало острых впечатлений при ошибочном указании каталога!
-i - Перед удалением каждого файла запрашивается подтверждение. Опция -i устраняет действие опции -f; она действует даже тогда, когда стандартный ввод не назначен на терминал.

SORT

sort test.txt

Несколько принципов, по которым команда sort linux сортирует строки:

Строки с цифрами размещаются выше других строк
Строки, начинающиеся с букв нижнего регистра размещаются выше
Сортировка выполняется в соответствии алфавиту
Строки сначала сортируются по алфавиту, а уже вторично по другим правилам.

Основные опции утилиты sort.
-b - не учитывать пробелы
-d - использовать для сортировки только буквы и цифры
-i - сортировать только по ASCII символах
-n - сортировка строк linux по числовому значению
-r - сортировать в обратном порядке
-c - проверить был ли отсортирован файл
-o - вывести результат в файл
-u - игнорировать повторяющиеся строки
-m - объединение ранее отсортированных файлов
-k - указать поле по которому нужно сортировать строки, если не задано, сортировка выполняется по всей строке.
-f - использовать в качестве разделителя полей ваш символ вместо пробела.

DIFF

diff [опции] файлы-или-директории

Основные опции команды diff:
-E - игнорировать изменения, связанные с добавлением символа табуляции в тексте.
-b - игнорировать изменения, связанные с добавлением пробелов.
-w - игнорировать изменения, связанные с добавлением пробелов и табуляции.
-B - игнорировать новые пустые строки.
-p (или —show-c-function) - показать название функции языка C, в которой найдены изменения.
-y (или —side-by-side) - отобразить результаты в две колонки.
-r - просматривать каталоги рекурсивно.
-X FILE - исключить из поиска файлы, имена которых совпадают с шаблонами в файле FILE.
-d (или —minimal) - попытаться найти как можно меньше изменений (то есть исключить ложные срабатывания).

CUT

cut -d : -f 1 /etc/passwd //Извлекает список пользователей текущей системы

Часто используемые опции командной строки:
-b, выбрать из файла только заданные байты согласно списку
-c, выбрать из файла заданные символы согласно списку
-f, выбирает только поля, перечисленные в списке. Разделителем по умолчанию служит TAB. Значение по умолчанию может быть переопределено с помощью опции -d.
-d, Позволяет задать разделитель полей. Как уже говорилось выше, значением по умолчанию является TAB, но эта опция позволяет переопределить его.

CD

cd опции папка_назначения

cd - //можете вернуться в предыдущую папку
cd .. // можно перейти в родительский каталог
cd ../../ //можете использовать несколько блоков с точками для перемещения на несколько уровней вверх
cd //если не передать папку, в которую нужно перейти, будет открыта домашняя папка

SUDO

Командная строка **sudo** может быть использована в следующих форматах:
sudo -h | -K | -k | -V
sudo -v [-AknS] [-g group] [-h host] [-p prompt] [-u user]
sudo -l [-AknS] [-g group] [-h host] [-p prompt] [-U user] [-u user] [command]
sudo [-AbEHknPS] [-C num] [-g group] [-h host] [-p prompt] [-u user] [VAR=value] [-i|-s] []
sudo -e [-AknS] [-C num] [-g group] [-h host] [-p prompt] [-u user] file

Параметры командной строки:
-A, --askpass - использовать вспомогательную программу для ввода пароля
-b, --background - выполнить команду в фоновом режиме
-C, --close-from=num - закрыть все дескрипторы файлов >= num
-E, --preserve-env - сохранить пользовательское окружение при выполнении команды
-e, --edit - редактировать файлы вместо выполнения команды
-g, --group=group - выполнить команду от имени или ID указанной группы
-H, --set-home - установить для переменной HOME домашний каталог указанного пользователя
-h, --help - показать справку и выйти
-h, --host=host - выполнить команду на узле (если поддерживается модулем)
-i, --login - запустить оболочку входа в систему от имени указанного пользователя; также можно задать команду
-K, --remove-timestamp - полностью удалить файл с timestamp
-k, --reset-timestamp - объявить недействительным файл timestamp
-l, --list - показать список прав пользователя или проверить заданную команду; в длинном формате используется дважды
-n, --non-interactive - автономный режим без вывода запросов пользователю
-P, --preserve-groups - сохранить вектор группы вместо установки целевой группы
-p, --prompt=prompt - использовать указанный запрос пароля
-S, --stdin - читать пароль из стандартного ввода
-s, --shell - запустить оболочку от имени указанного пользователя; также можно задать команду
-U, --other-user=user - в режиме списка показывать права пользователя
-u, --user=user - выполнить команду (или редактировать файл) от имени или ID указанного пользователя
-V, --version - показать сведения о версии и выйти
-v, --validate - обновить временную метку пользователя без выполнения команды
-- - прекратить обработку аргументов командной строки

sudo apt-get update //Команда update используется для повторной синхронизации индекса файлов пакета из своих источников, указанных в файле /etc/apt/sources.list.

sudo apt-get upgrade //Команда upgrade используется для обновления всех установленных пакетов программного обеспечения в системе. При любых обстоятельствах уже установленные пакеты не будут удалены, или пакеты, которые еще не установлены, не будут извлечены и установлены для удовлетворения зависимостей обновления.

sudo apt-get install //Команда install отслеживает один или несколько пакетов для установки или модернизации