МИНОБРНАУКИ РОССИИ

ФГБОУ ВО «Ижевский государственный технический университет имени М.Т.Калашникова»

В.Г. Тарасов

Методические указания

к выполнению лабораторной работы № 2

на тему «Настройка сетевого подключения в системе FreeBSD и работа с утилитой Make»

по дисциплине «Операционные системы»

|  |
| --- |
| Ижевск 2018  Рег. номер |

Методические рекомендации по выполнению лабораторной работы составлены в соответствии с рабочей программой учебной дисциплины, разработанной на основе Федерального государственного образовательного стандарта по направлению 09.03.04 «Программная инженерия» профиль «Разработка программно-информационных систем» программа подготовки «Академический бакалавриат» при изучении дисциплины «Операционные системы»

Рецензент: *П.В. Лекомцев, канд. техн. наук, доцент*

Составитель: *В.Г.Тарасов, канд. техн. наук, профессор*

Рекомендовано Советом института ИВТ для использования в учебном процессе в качестве учебно-методических материалов для студентов, обучающихся по направлению09.03.04 «Программная инженерия» профиль «Разработка программно-информационных систем» программа подготовки «Академический бакалавриат» при изучении дисциплины «Операционные системы»(протокол № \_\_ от «\_\_\_» марта 2018 г.)

СОДЕРЖАНИЕ

1.Общие положения4

2.Порядок выполнения работ4

3.Настройка сети5

4.Пакеты. Описание и установка11

5.Установка файлового менеджера Midnight Commander 12

6.Работа с утилитой make13

7.Использование отладчика LLDB для отладки программы20

8.Еще немного о make-файлах21

9.Лабораторное задание23

10.Содержание отчета23

11.Контрольные вопросы23

Список рекомендуемой литература24

Приложение 1. Установка через пакеты25

Приложение 2. Файловый менеджер Midnight Commander27

ЛАБОРАТОРНОЕ ЗАНЯТИЕ 2. Установка пакетов-портов в ОС FreeBSD

ЦЕЛЬ РАБОТЫ: изучение процедур и технологии настройки сетевых сервисов операционной системы семейства Linux, установки дополнительных пакетов и исследование сред виртуального компьютера.

1.ОБЩИЕ ПОЛОЖЕНИЯ

Операционные системы семейства Linux являются бесплатными и подпадают под так называемую GNU General Public License (универсальную общественную лицензию GNU), или GPL [1]. ОС Linux считается одной из наиболее мощных и надежных сетевых систем в мире. Linux поддерживает два главных сетевых протокола для Unix-систем: TCP/IP и UUPC. Протокол TCP/IP (Transmission Control Protocol/Internet Protocol) является набором сетевых парадигм, позволяющим компьютерным системам по всему миру связываться по единой сети, известной как Internet. С помощью Linux, TCP/IP и подключения к сети вы можете общаться с пользователями и машинами в Интернете через Web, электронную почту, телеконференции Usenet, передачу файлов по FTP и другие службы.

В большинстве сетей TCP/IP в качестве физического транспорта передачи данных используется Ethernet. Linux поддерживает многие распространенные сетевые карты Ethernet и интерфейсы для персональных и переносных компьютеров, а также Fast Ethernet, Gigabit Ethernet, ATM, ISDN, беспроводные адаптеры LAN, Token Ring, пакетные радиопередачи и ряд высокопроизводительных сетевых интерфейсов.

В лабораторной работе предполагается произвести сетевые настройки на виртуальной машине, которые позволят открыть доступ в сеть Интернет через сервер лаборатории. После подключения машины к сети станет возможна установка пакетов, которые позволяют установить другие дополнительные подсистемы, в частности, файловый менеджер mc.

2.ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

Лабораторная работа выполняется фронтально в учебно-исследовательской лаборатории кафедры программного обеспечения. На рабочих станциях установлено программное обеспечение Oracle VM VirtualBox. Данная лабораторная работа является продолжением первой: считаем, что у нас функционирует виртуальная машина (ВМ) с ядром Linux, текстовым редакторам ee и командным интерпретатором bash. Установлены также и драйверы основных устройств ВМ – для нас сейчас важен сетевой адаптер. В учебно-исследовательской лаборатории соединения компьютеров произведены с помощью сетевых карт Ethernet.

Сами задания будут представлены в следующих пунктах.

3. НАСТРОЙКА СЕТИ (**Задание 1**)

Настроить сетевое подключение для операционной системы FreeBSD 10.1 (или выше), которая работает на виртуальной машине VirtualBox 4.3.26 (или выше). В результате выполнения работы мы должны получить виртуальную машину с операционной системой FreeBSD, которая имеет выход в интернет.

1. Состояние системы до настройки сетевого подключения

Запускаем виртуальную машину с операционной системой FreeBSD, которую мы создали в ходе выполнения предыдущей лабораторной работы, авторизуемся в системе как созданный нами пользователь student и наделяем его root-правами (команда ***su***). Пока что все действия нам знакомы.

Для настройки сети нам понадобится отредактировать два файла, расположенных в папке /etc. Эта папка содержит в себе основные файлы конфигурации системы. Нужные нам файлы называются –rc.conf и resolv.conf.

Файл rc.conf содержит в себе основные настройки системы. Если открыть его с помощью команды ***cat***, то можно увидеть, что внутри на данный момент почти ничего нет, так как во время установки системы мы не поставили ни одной службы (рис.3.1). Все, что хранит в данный момент файл rc.conf - это хостовое имя нашей системы **freebsd12** и информацию о том, что система записи рабочей памяти системы не записывается (переменная **dumpdev = "NO"**).



Рис.3.1

Второй же нужный нам файл resolv.conf на данный момент не существует, вы можете убедиться в этом сами, попытавшись вызвать его командой ***cat***.

1. Описание настройки

Чтобы начать настройку сети, воспользуемся командой ***bsdconfig***. Эта команда вызовет следующее окно ( рис. 3.2).

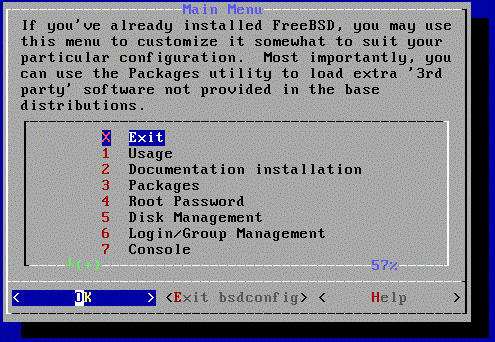


Рис.3.2

В открывшемся меню выберем пункт «Networking Management» (Управление сетями). Чтобы его увидеть, нужно пролистать список пунктов вниз или нажать на клавиатуре клавишу A(Ф). В появившемся подменю (рис.3.3) выбираем пункт «Network Interfaces» (Сетевые интерфейсы).

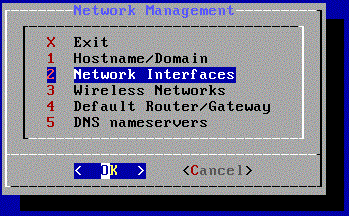


Рис.3.3

Видим уже знакомое нам по установке системы окно (рис.3.4). Оно показывает нам список доступных сетевых интерфейсов ( в нашем случае – это em0). Здесь нам ничего не нужно менять, нажимаем «OK» и переходим к следующему этапу – рис. 3.5.

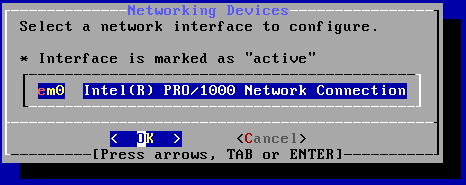


Рис.3.4

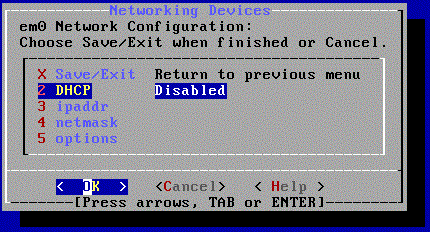


Рис.3.5

В меню настройки выбираем пункт «DHCP» для активации DHCP-клиента. После чего мы увидим список настроек, полученных от DHCP-сервера (рис.3.6).

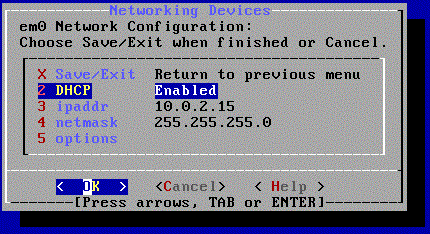


Рис.3.6

Далее выбираем пункт «Save/Exit» - сохраняем настройки и выходим в предыдущее меню. Во всех последующих окнах так же выбираем «Exit» и возвращаемся обратно к командной строке.

1. Состояние системы после настройки сетевого подключения

После проведения настройки система должна была сохранить конфигурацию сетевого интерфейса файлах rc.conf и resolv.conf.

В файле rc.confпоявилась переменная **ifconfig\_em0 = "DHCP"**(рис.3.7).



Рис.3.7

Созданный же только что файл resolv.conf теперь содержит в себе несколько переменных типа **nameserver** (рис.3.8).



Рис.3.8

1. Описание конфигурационных файлов

Добавившаяся в файл rc.confстрочка **ifconfig\_em0 = "DHCP"** сообщает системе инициализации о том, что сетевой интерфейс **em0** необходимо настраивать, используя протокол DHCP.

DHCP, или Dynamic Host Configuration Protocol (Протокол Динамической Конфигурации Хостов), описывает порядок, по которому система может подключиться к сети и получить необходимую информацию для работы в ней (IP-адрес, маску подсети, DNS-сервер).

Что касается файла resolv.conf, то этот файл содержит информацию, позволяющую компьютеру преобразовывать буквенно-цифровые доменные имена в цифровые IP-адреса. На рис. 3.8 можно увидеть, что на данный момент он содержит переменные типа **nameserver**, которые указывают на IP-адрес DNS-сервера.

1. Проверка настроек сетевого подключения

Для проверки настройки сетевого подключения необходимо выполнить команду ***ifconfig***. Пример вывода данной команды представлен на рис.3.9.

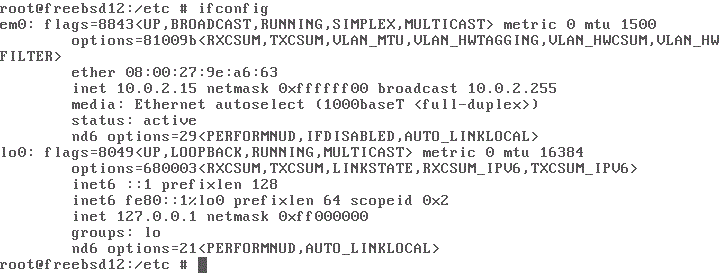


Рис.3.9

Поле **inet** содержит IP-адрес интерфейса, а поле **netmask** – маску подсети. Если они заполнены, то значит, все настройки выполнены правильно.

Для проверки доступа в интернет можно воспользоваться командой ***ping***. Эта команда используется для проверки соединения с применением ICMP-протокола и является одним из основных диагностических средств в сетях TCP/IP.

ICMP (Internet Control Message Protocol) или протокол межсетевых управляющих сообщений используется для передачи сообщений об ошибках и других исключительных ситуациях, возникших при передаче данных, например, запрашиваемая услуга недоступна, или хост, или [маршрутизатор](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%80%D1%88%D1%80%D1%83%D1%82%D0%B8%D0%B7%D0%B0%D1%82%D0%BE%D1%80) не отвечают. Также на ICMP возлагаются некоторые сервисные функции.

Выполним команду ***ping google.com*.** На экране будет видна последовательная передача данных между нашим компьютером и вызванным сайтом. Остановить процесс можно точно так же, как и другие – сочетанием клавиш ctrl+C (рис.3.10).

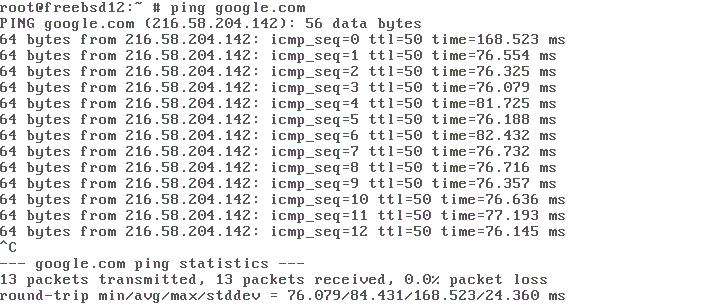


Рис.3.10

Вывод команды вернул время затраченное на передачу пакета т.е. пакет был доставлен до адреса **google.com**.Это говорит нам о том, что настройка сети проведена верно.

Для того, чтобы окончательно убедится, что проблем с подключением нет, перезагрузим машину и проверим доступность интернета еще раз. Результаты должны быть практически теми же. Для разнообразия можно попробовать подключится по адресу **ya.ru**  и сравнить результаты.

1. Анализ полученных результатов

Виртуальная машина с FreeBSD получает сетевые настройки от встроенного в VirtualBox DHCP-сервера. Доступ к интернету из виртуальной машины осуществляется с помощью механизма NAT (настройка по умолчанию).

NAT (Network Address Translation — «преобразование сетевых адресов») — это механизм в сетях TCP/IP, позволяющий преобразовывать IP-адреса транзитных пакетов.

Доступа к интернету из виртуальной машины не будет, если хост (машина, на которой работает VirtualBox) сам не имеет подключения к интернету.

Возможные варианты подключений к сети интернет представлены на рис.3.11.

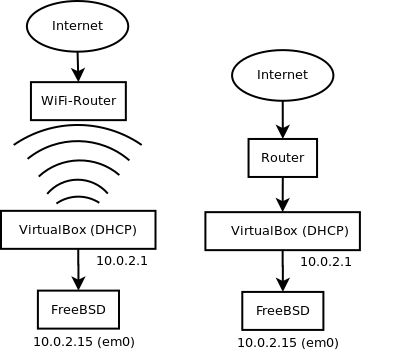


Рис.3.11

Внутри VirtualBox работает DHCP-сервер, который по протоколу DHCP выдает виртуальным машинам необходимые настройки для сетевого интерфейса (IP-адрес, маска подсети и DNS-сервер). Адрес DHCP-сервера VirtualBox (10.0.2.1), адрес виртуальной машины с FreeBSD (10.0.2.15). VirtualBox получает необходимые настройки от хоста, который, в свою очередь, получает их от другого DHCP-сервера, который обычно встроен в роутер, используя проводное или беспроводное соединение.

Таким образом, после выполнения описанных выше настроек мы имеем доступ в интернет, используя хост, который может быть подключен к интернету через витую пару или через WiFi.

4. ПАКЕТЫ. ОПИСАНИЕ И УСТАНОВКА (**Задание 2**)

Вместе с FreeBSD поставляется богатый набор системных утилит. Однако для выполнения какой-то реальной работы очень скоро возникает необходимость в установке дополнительных приложений. FreeBSD дает две взаимодополняющих технологии для установки программного обеспечения сторонних разработчиков: порты (для установки из исходных кодов) и пакеты (для установки из бинарных файлов). На сегодняшний день пакеты являются более быстрым и простым способом установки, чем порты, поэтому в лабораторных работах рекомендуется использовать именно их.

В последних версиях FreeBSD была добавлена утилита pkgng, значительно упрощающая установку пакетов. Обычно она устанавливается автоматически. Проверить это можно командой ***pkg*** (без параметров): пользователю будет предложена установка утилиты, если это еще не произошло (рис.4.1). Установка утилиты требует наличия у пользователя **root**-прав.

Для установки через пакеты достаточно ввести команду ***pkg install******pakege’s\_name***. После этого будет установлен нужный пакет. В некоторых случаях будут автоматически подгружены и установлены дополнительные пакеты, которые требуются для работы первого (требуется подключение к сети). Если пакет больше не используется, его можно удалить командой ***pkg delete pakege’s\_name***.

Этих 2 команд должно хватить для выполнения лабораторных работ.

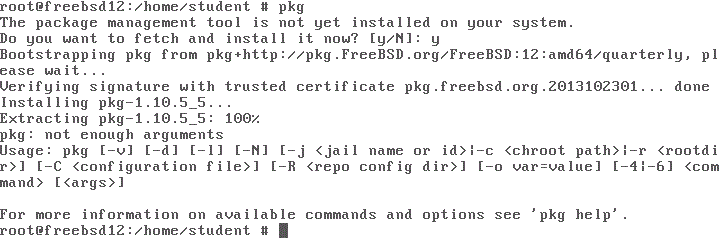


Рис.4.1

5. УСТАНОВКА ФАЙЛОВОГО МЕНЕДЖЕРА MIDNIGHT COMMANDER **(Задание 3)**

Для установки файлового менеджера **mc (Midnignt Commander)** выполните команду ***pkg install mc***.

Вскоре после начала загрузки, вам будет задан вопрос о продолжении установки, вы должны согласиться (т.е. ввести символ 'y'). На рис. 5.1. представлен процесс установки. Запрос может повториться несколько раз.

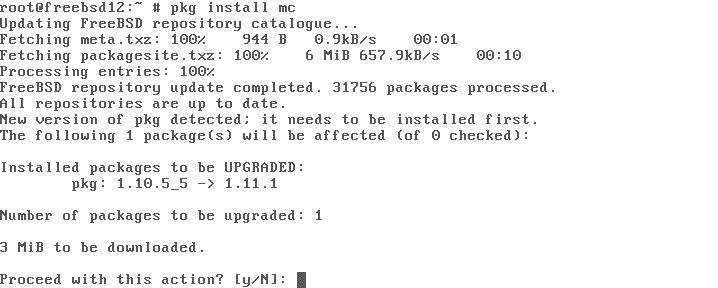


Рис.5.1

Теперь, когда установка завершена, запускаем mc командой ***mc***. Результат можно увидеть на рис.5.2.

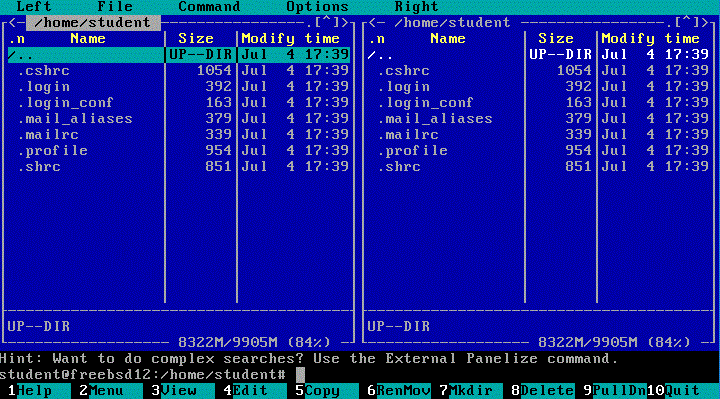


Рис.5.2

До этого мы работали с текстовым редактором EE, но теперь можем использовать mc для тех же целей. Подробнее о файловом менеджере mc смотрите в Приложении 2.

6.РАБОТА С УТИЛИТОЙ MAKE (**Задание 4)**

Начнем с того, что вообще такое эта утилита **make** и для чего она может нам пригодиться.

Итак, в прошлой лабораторной работе мы научились создавать простые программы, состоящие всего из одного файла с исходным кодом. Вроде бы ничего сложного: создаем файл -> записываем туда программу -> получаем объектный файл -> собираем из объектного файла загрузочный. И вот перед нами четыре простых пункта, который не так уж тяжело и долго выполнять. Но что если у нас несколько файлов с исходным кодом? Конечно, допустим, что можно прописать компиляцию вручную для двух, и даже для двадцати файлов, хоть это и будет ужасно долго и нудно. Вот только файлов может быть и сто, и тысяча, да и работать вы можете над ними не одни. К тому же, когда в разработке находится не маленький «hello, world», а огромный проект, состоящий из множества файлов, которым занимается ничуть не меньшее множество разработчиков, да еще и в разных местах и в разное время, то и изменяться все эти модули будут тоже не одновременно. Пересобирать же весь проект каждый раз, когда в 103-ем файле добавили один новый символ, будет ужасно как для людей, так и для техники. Вот тут-то нам и пригодится утилита **make**.

**Make** - это программа, которая обычно используется при разработке больших программ, состоящих из нескольких файлов. Она следит за тем, какие объектные файлы зависят от каких исходных файлов и файлов заголовков, и при вызове выполняет минимальное количество перекомпиляций для обновления целевого файла. Но чтобы эта магическая помощница сделала свое дело, нам нужно будет понять, каким именно правилам нужно следовать. Make читает файл, называемый *make-файлом*, указывающий, как различные файлы зависят друг от друга, и определяет, каким файлам требуются перекомпиляция, а каким нет. Например, правило может говорить нечто вроде "если hello.o старее, чем hello.c, то это значит, что кто-то изменил hello.c, поэтому этот файл нужно перекомпилировать". В make-файле также имеются правила, указывающие программе make, *как именно* перекомпилировать файл с исходным текстом, делая эту утилиту гораздо более мощным инструментом.

Make-файлы обычно располагаются в том же самом каталоге, что и исходные файлы, к которым они имеют отношение, и могут называться makefile, Makefile или MAKEFILE. Большинство программистов используют имя Makefile, так как при этом оно будет помещено в начало списка файлов каталога, где его легко увидеть.

Для работы с утилитой make сначала ее нужно установить. Воспользуемся для этого уже знакомыми нам пакетами и выполним команду ***pkg install gmake***. Это не займет много времени.

Теперь можно приступить к изучению работы самой утилиты. Для этого давайте выйдем из пользователя root обратно в student при помощи команды exit. Сейчас мы снова вернулись в home, где уже лежат файлы, созданные нами в предыдущей лабораторной работе. Рассмотрим принцип работы программы **make** на их примере.

Для начала, чтобы программа make приступила к выполнению своей работы, нужно вызвать ее в командной строке. Для этого используется команда ***make***. Если запустить ее, то программа попытается найти первый попавшийся make-файл в текущем каталоге и выполнить инструкции из него. В случае, если вам нужен не любой, а конкретный make-файл, то к нему можно обратится по ключу ***–f***. Например, так ***make –f MyMakefile****.* Мы же пока не создавали никаких make-файлов, так что при вызове, команда ***make*** не найдет никаких целей и завершит свое выполнение.

Теперь давайте поговорим о структуре make-файла. В этом файле есть три основных элемента:

ЦЕЛЬ\_1: ЗАВИСИМОСТИ\_1

КОМАНДА\_1

Обратите внимание, перед командой **всегда** должен стоять символ табуляции.

Подобных связок внутри make-файла может быть много, но при вызове автоматически будет выполняться только самая первая. Чтобы выполнить остальные, нужно либо связать их внутри файла, либо вызвать отдельно.

Давайте сделаем самый простой make-файл из уже имеющегося у нас кода. Так как мы уже установили файловый менеджер **mc**, то вместо **ee** работать будем в нем. Вызовем команду ***mc*** в командной строке и убедимся, что мы находимся в домашнем каталоге пользователя student. Создать новый файл можно сочетанием клавиш Shift+F4. Назовем файл просто makefile. Запишем в него компиляцию нашего hello world:

test: test.c

clang++ -o test test.c

Сохраним(F2) и вызовем команду ***make***. Но сначала удалим уже имеющийся у нас файл test. Для того, чтобы посмотреть на командную строку, не выходя из mс, можно воспользоваться сочетанием клавиш Ctrl+O. Каждый раз при открытии mc, лог командной строки очищается, поэтому, чтобы увидеть результат, стоит вызвать команду уже после перехода.

Если вы все сделали правильно, то файл test будет создан заново и мы сможем его запустить.

Это был очень простой make-файл. Давайте рассмотрим более сложный на примере этих абстрактных целей.

Пример make-файла с несколькими целями:

prog: head.o tail.o # цель ***prog*** зависит от

#файлов *head.o* и *tail.o*

clang++ -o prog head.o tail.o # сообщает **make**, как

#достичь цели *program*

@echo Program done. # извещает о завершении

# процесса компиляции

head.o: head.c # цель head.o зависит от

#файла head.c

clang++ –c head.c

tail.o: tail.c # цель tail.o зависит от

# файла tail.c

clang++ –c tail.c

Обратите внимание, что при указании наборе команды (в данном примере это вызовы **clang++** и **echo**) используются символы табуляции без пробелов. Цель prog использует зависимости head.o и tail.o, поэтому, когда make начнет выполнение этой цели, то в случае, если их нет, ей придется отправится к целям head.o и tail.o и выполнить сначала их.

Теперь, когда мы разобрались с основами, перейдем к более сложному заданию:

* 1. Написать простую программу с использованием библиотеки **ncurses**.

Библиотека **ncurses** служит для создания программ, работающих с окнами, управления курсором в текстовом режиме, задания цвета символов и фона и т.д. Для демонстрации возможностей библиотеки в рассмотренном ниже примере в середину пустого экрана выводится сообщение "Hello, World!". После нажатия на любую клавишу оно исчезает, и программа заканчивает работу. Моя программа выглядит так.

#include <curses.h>

#include <stdio.h>

#include <string.h>

int main()

{

char s[]="Hello, World!";

int ls=strlen(s);

initscr(); cbreak(); noecho();

mvprintw( LINES/2, COLS/2-ls/2, "%s", s );

getch();

echo();

nocbreak();

endwin();

return 0;

}

*Для справки.* Функции из библиотеки ncurses

nocbreak()/cbreak() -- включение/выключение буферизации ввода.

nodelay(stdscr, bool Value) – включение/выключение режима блокировки getch.

При включении режима nodelay getch() вместо того чтобы ожидать ввода символа вернет ERR.

move(int y, int x) – перемещает курсор в позицию (x,y)

printw() -- аналог printf для ncurses. Есть версия mvprintw которая первыми двумя аргументами берет y и x и перемещает курсор туда перед выводом.

int usleep(useconds\_t microseconds) – подождать определенное время (в микросекундах).

int getch() -- Функция читает символ со стандартного ввода. В случае если включена буферизация ввода, символ будет получен только после нажатия пользователем enter, иначе символ будет получен сразу.

В случае если включено эхо (echo() ), символ будет выведен на экран, иначе нет.

В случае если включен режим nodelay, вызов функции getch() не будет дожидаться ввода символа. Вместо этого в случае отсутствия символа в буфере он вернет значение ERR.

Ниже приведен пример программы, которая каждые полсекунды опрашивает клавиатурный буфер. Если пользователь нажимал на клавиши, то печатается сообщение **"Key was pressed".** Если нажатий в последние полсекунды не было, то печатается **"No key was pressed!"**. Получите у преподавателя задание на составление подобной «экранной» программы.

#include <stdio.h>

#include <curses.h>

#include <unistd.h>

int main(void)

{ int c=32, c1, x;

initscr();

scrollok(stdscr, true); //включить прокрутку экрана

nodelay(stdscr,true); //включить неблокирующий режим для getch()

nocbreak(); //выключить буферизацию

while (c != 27)

{ if ( (x=getch())!=ERR ) //if keypressed then readkey...

{ addstr(" Key was pressed\n"); //addstr – самая простая функция вывода. // выводит одну строку.

do if ( x == 27 ) c=(char)x;

while ((x=getch())!=ERR);

}

else addstr("No key was pressed!\n");

usleep(500000); /0.5 секунды!

} // while

nodelay(stdscr,false); //вернуть getch() на место

return 0;

}

* 1. Стандартные цели для makefile.

Для того чтобы быть удобным множеству пользователей, ваш makefile должен поддерживать некоторые стандартные цели.

all – компиляция программы, является стандартной целью по умолчанию. При вызове ее можно явно не указывать.

clean – очистить каталог от всех файлов, полученных в результате компиляции.

install – копирование бинарных файлов программы в пути по умолчанию.

uninstall/deinstall – удаление всех созданных нами с помощью цели all бинарных файлов из системы.

* 1. Написание цели all.

Рассмотрим подробно процесс компиляции. Сначала из каждого исходного файла (с расширением .c) создается объектный файл (с расширением .o). Затем из одного или нескольких o файлов собирается исполняемый файл. Для того чтобы программа могла использовать библиотеку ncurses, надо написать -lncurses, чтобы линкер слинковал с ней.

all: compile

compile: chello

chello: curse.o

clang++ curse.o -lncurses -o chello

Для получения объектного файла из исходного добавляется следующая цель

curse.o: curse.c

clang++ -c curse.c

В этом примере файлы с исходным текстом и объектным кодом называются curse, а исполняемый - chello.

* 1. Написание цели clean.

Приведенный пример цели должен удалить исполняемый файл и все файлы типа .o.

clean:

rm \*.o chello

* 1. Написание целей install и uninstall/deinstall.

Эта цель должна скопировать исполняемый файл в /usr/local/bin (/usr означает, что программа не системная, local – собрана на этой системе, а не поставлена пакетом, bin – исполняемый файл).

install: chello

cp chello /usr/local/bin/chello

deinstall uninstall:

rm /usr/local/bin/chello

Итоговый makefile выглядит так:

all: compile

compile: chello

chello: curse.o

clang++ curse.o -lncurses -o chello

curse.o: curse.c

clang++ -c curse.c

clean:

rm \*.o chello

install: chello

cp chello /usr/local/bin/chello

deinstall uninstall:

rm /usr/local/bin/chello

Выполнять цели можно так:

make all install - для компиляции и установки программы в нужную   
директорию;

make clean - для удаления объектных и исполняемых файлов.

*Продемонстрируйте* возможности Вашего makefile.

Много полезных и удобных приемов можно узнать в статье: <http://www.linux.org.ru/books/GNU/Gmake.htm>

7. ИСПОЛЬЗОВАНИЕ ОТЛАДЧИКА LLDB ДЛЯ ОТЛАДКИ ПРОГРАММ (**Задание 6**)

Предположим, написана программа bug.c, например такая:

#include <stdio.h>

int main()

{

int i;

for(i=10;i>=0;--i)

{

printf("%d ",10/i);

}

puts("\n");

return 0;

}

При запуске программа выдает ошибку: Floating point exception.

Для отладки при помощи lldb надо проделать следующие действия:

1. Собрать программу с ключом отладки -g:

clang -g bug.c -o bug

1. Запустить программу под отладчиком: lldb bug.

Теперь мы находимся в отладчике lldb и можем заняться отладкой нашей программы. Покинуть отладчик можно при помощи команды ***exit***.

Для того чтобы просмотреть программу без подключённых библиотек, можно использовать команду ***list***. Если задать вместе с ней номер строки, то отладчик покажет код, начиная с этой строки. Например, ***list 5***. Отладчик lldb может распознавать команды по первым символам, так что команда ***list 5*** будет равнозначна команде ***l 5***.

Пример вывода команды ***list*** представлен на рис.7.1.

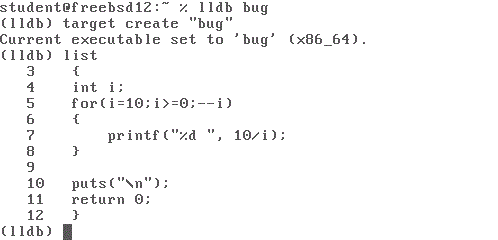


Рис. 7.1. Пример вывода отладчика

Следующий шаг – установить точку остановки. Для этого служит команда break с аргументом – номером строки (рис. 7.2).

C:\Users\1\Desktop\ОС\Изображения\Лаб2\рис 7.2.GIF

Рис.7.2. Пример работы команды break

Теперь можно и запустить программу. Она дойдет до строки с точкой останова и снова выведет приглашение. Для запуска служит команда run.

Чтобы сделать шаг по программе, используется команда next.

В случае, когда следующий оператор содержит вызов функции и нам необходимо войти внутрь этой функции с целью изучения ее поведения, используется команда step.

Чтобы вывести содержимое переменной, надо использовать print <имя>.

C:\Users\1\Desktop\ОС\Изображения\Лаб2\рис 7.3.GIF

Просматривать по шагам циклы – скучное занятие. Чтобы запустить программу на дальнейшее выполнение, нужно ввести continue или c.

Program received signal SIGFPE, Arithmetic exception.

0x080483b4 in main () at bug.c:8

8 printf("%d ",10/i);

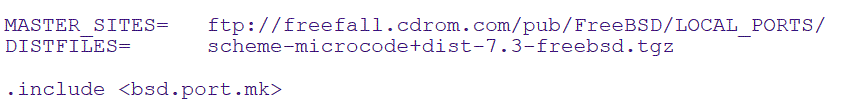
Ошибка почти найдена! Если посмотреть на содержимое i, то мы увидим 0, на который мы и пытаемся поделить (!).

8.ЕЩЕ НЕМНОГО О MAKE-ФАЙЛАХ (**Задание 7**)

Ранее мы уже разобрали, как работает утилита make в основных случаях. Вот только это не все ее возможности. На самом деле make-файлы могут быть достаточно сложными в создании. Например, когда они работают с портами.

Мы уже познакомились с таким способом установки как пакеты, где вскользь упомянули о существовании портов. Несомненно, пакеты позволяют куда быстрее и проще получить и установить интересующие нас программы, но при этом они во многом ограничены. Порты же - это сценарии командной оболочки, скачивающие исходный код программы и компилирующие из него бинарный файл. Установка из портов занимает весьма продолжительное время (зависит от скорости сети и мощности процессора вашего компьютера), но позволяет гибко настраивать и оптимизировать любое ПО. А сценарии – это и есть уже знакомые нам make-файлы. Makefile каждого порта автоматически сгружает исходный код приложения с локального диска или CDROM либо по [ftp](http://www.freebsd.org/cgi/man.cgi?query=ftp), распаковывает его в вашей системе, прикладывает патчи и компилирует. Если всё проходит гладко, набор команды make install приводит к установке приложения и регистрации его в системе.

И так как обычно написание подобных make-файлов сложно и занимает много времени, то многие системы (такие, как FreeBSD), поставляются с несколькими очень мощными в составе системы. Одним очень хорошим примером является система портов FreeBSD. Вот основная часть типичного Makefile порта:



Теперь, если мы перейдем в каталог с этим портом и наберем команду **make**, то случится вот что:

1. Проводится проверка на наличие исходного кода этого порта в системе.
2. Если его нет, осуществляется FTP-соединение к URL, указанному в MASTER\_SITES, для сгрузки исходного текста.
3. Вычисляется контрольная сумма исходных текстов и сравнивается с известной контрольной суммой оригинальной копии исходных текстов. Это делается для того, чтобы проверить целостность исходных текстов после перекачки.
4. Делаются все изменения, требуемые для того, чтобы исходный код работал во FreeBSD--этот процесс носит название *патчинг* (patching).
5. Выполняются все настройки, требуемые для исходных текстов. (Многие дистрибутивы Unix-программ пытаются определить, на какой версии Unix они компилируются и какие дополнительные возможности Unix имеются--эта информация информация задается в сценариях портов FreeBSD).
6. Компилируется исходный код программы. В частности, мы меняем каталог на тот, в который были распакованы исходные тексты и выполняем команду make--в собственном make-файле программы имеется необходимая для построения программы информация.
7. Теперь у нас есть откомпилированная версия программы. Если мы хотим, то можем ее протестировать; когда мы останемся довольными ей, то можем выдать команду ***make install***. При этом программа и все необходимые для работы файлы будут скопированы в правильные каталоги; в базе данных о пакетах будет сделана запись, чтобы порт мог быть позже с легкостью удален, если он нам не понравится.

Теперь, я думаю, вы согласитесь, что для сценария из четырех строк это выглядит весьма впечатляюще!

Секрет заключен в последней строке, которая указывает программе make на включение системного make-файла с именем bsd.port.mk. Легко пропустить эту строчку, но именно в ней содержатся все эти хитрости--кто-то написал make-файл, указывающий программе make на выполнение всех вышеописанных действий (плюс еще многих мною не описанных, включая обработку ошибок, которые могут возникнуть) и любой может получить доступ к нему, просто добавив одну строчку в собственный make-файл!

Получите у преподавателя задание, связанное с созданием простейшего порта при помощи make-файлов.

9. ЛАБОРАТОРНОЕ ЗАДАНИЕ

Получить от преподавателя задание на разработку алгоритма и программы на языке программирования С с использованием возможностей библиотеки ncurses.

Подготовить наборы тестовых начальных значений и соответствующих результатов, необходимых для проверки правильности разработанной программы.

Используя представленные в описании примеры работы с текстовыми редакторами и справочные материалы, произвести необходимые вычисления на виртуальной машине под управлением установленной ОС, используя команды процессора [bash](http://www.FreeBSD.org/cgi/man.cgi?query=csh&sektion=1).

Провести вычисления с использованием средств автоматизации построения программных проектов – утилиты make.

Создать простейший порт, используя возможности утилиты make.

10. СОДЕРЖАНИЕ ОТЧЕТА

1. Цель работы.
2. Текст варианта задания.

Расчет тестовых начальных значений и соответствующих результатов, необходимых для проверки правильности разработанной программы.

1. Текст программы с подробными комментариями.
2. Результаты экспериментирования на ЭВМ.
3. Выводы.

11. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие сетевые настройки позволяет произвести сервис **DHCP** (Dynamic Host Configuration Protocol - протокол динамической конфигурации узла)?
2. Почему стоит использовать файловый менеджер mc?
3. Какие возможности предоставляет программисту библиотека ncurses?
4. Какова структура makefile?
5. Какие существуют команды для проведения отладки программ с помощью программы-отладчика lldb? Какие еще вы знаете низкоуровневые отладчики?

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. Уэлш М., Далхаймер М.К., Кауфман Л. Запускаем Linux. – Пер. с англ. – СПб: Символ-Плюс, 2001.
2. Таймэн Б. FreeBSD 6. Полное руководство. – М.: Вильямс, 2007.
3. <ftp://ftp.FreeBSD.org/pub/FreeBSD/doc/>.
4. <http://www.oszone.net>.
5. http://www.linux.org.ru/books/GNU/Gmake.htm.

ПРИЛОЖЕНИЕ 1

УСТАНОВКА ЧЕРЕЗ ПАКЕТЫ

Система управления пакетами - один из существенных компонентов любой операционной системы. Она включает средства установки, учета, обновления и удаления программного обеспечения, не обязательно входящего в базовую поставку собственно ОС. Традиционно такие системы разделяются на две группы - для работы с прекомпилированными пакетами и для сборки бинарных пакетов из исходных текстов. Впрочем, с течением времени граница между ними все более стирается.

Пакет, как правило, представляет собой простой tar-архив, сжатый утилитой gzip или bzip2, то есть тарбалл (специфичный для Linux/UNIX систем формат архива .tar.gz или .tgz) вида pkg\_name-pkg\_version.tgz (pkg\_name-pkg\_version.tbz), который можно просмотреть обычной командой вроде

**# tar tzvf pkg\_name-pkg\_version.tgz**

или

**# tar tjvf pkg\_name-pkg\_version.tbz**

cоответственно. Процедура эта показывает, что в состав пакета входят, как правило, следующие компоненты:

* скомпилированный и слинкованный (обычно динамически) исполняемый файл (или файлы), помещенный в каталог bin или, для программ системного назначения, sbin (относительно корня архива);
* необходимые конфигурационные файлы (каталог etc);
* документацию в виде man-страницы или серии man-страниц (один из каталогов man/man#);
* разного рода разделяемые файлы (подкаталоги каталога share), в частности - документацию в отличных от man-страниц форматах, если таковая предусмотрена разработчиком пакета (но не сборщиком - в BSD-системах обязательными являются только man-pages);
* всякого рода библиотечные и заголовочные файлы (каталоги lib, include и так далее), если таковые присутствуют в пакете;
* специальные файлы, содержащие версию, описание, принадлежность пакета, зависимости его от других, вспомогательные скрипты для настройки/удаления пакета.

Макет команды для запуска установки пакета выглядит так:

**# pkg install имя\_пакета**

Данная команда запустит установку указанного пакета. Сначала система проверит обновления, обратившись к пакетному репозиторию. Адрес используемого по умолчанию репозитория указан в /usr/local/etc/pkg.conf. Если репозиторий уже полностью обновлен, то система будет искать указанный пакет. Если пакет найден, на экране появятся его зависимости и запрос на подтверждение.

После этого перечисленные на экране пакеты будут загружены и установлены. Некоторые пакеты предоставят важную послеинсталляционную информацию или инструкции по использованию приложения; обязательно выполните все появившиеся инструкции.

Основным же преимуществом пакетной системы является, в первую очередь, автоматическое удовлетворение зависимостей пакета, то есть все пакеты, от которых зависит устанавливаемый пользователем пакет, инсталлируются автоматически.

ПРИЛОЖЕНИЕ 2

ФАЙЛОВЫЙ МЕНЕДЖЕР MIDNIGHT COMMANDER

После установки и запуска mc перед нами появятся две ярко-синие панели с указателем светло-бирюзового цвета (рис. п. 2.1), который можно перемещать при помощи кнопок «вверх» и «вниз» на клавиатуре. Чтобы открыть выбранную директорию или файл, нужно воспользоваться клавишей «enter». Так же можно выполнить некоторые другие действия, используя функциональные клавиши (копировать, переместить, удалить и т.д.). Так же в mc существуют команды и сочетания кнопок (горячие клавиши), знание которых делает работы в файловом менеджере более эффективной и удобной.

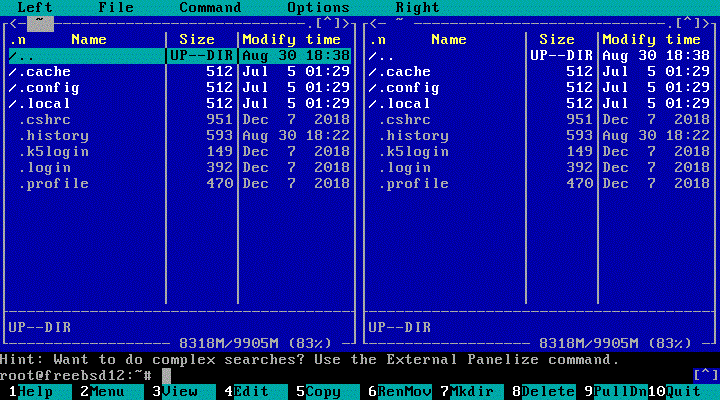


Рис. п.2.1

Функциональные клавиши расположены в нижней строке mc. Вот их полный список:

F1 – всплывет серое окно с подсказками;

F2 – вызов пользовательского меню;

F3 – просмотр содержимого файла;

F4 – редактирование содержимого файла (в сочетании с клавишей «shift» - создание нового файла);

F5 – копирование файла/директории;

F6 – переноси или переименование файла/директории;

F7 – создание новой директории;

F8 – удаление файла/директории;

F9 – переход в основное меню (верхняя строка mc), вернуться обратно можно при помощи клавиши «esc»;

F10 - выход (завершение работы файлового менеджера).

Так же можно использовать «горячие клавиши». Вот некоторые из них:

TAB, Ctrl+I –переход между панелями

Insert, Ctrl+T – выделение файла/директории для каких либо операций над ними (повторное нажатие на выделенном объекте снимает выделение)

Ctrl+O – убрать панели с экрана (удобно для выполнения в консоли какой-либо команды, не выходя при этом из mc. Повторное нажатие вновь покажет панели)

Ctrl+Space – подсчет размера каталога

Ctrl+U – поменять панели местами

Ctrl+S. Alt+S – быстрый поиск файлов (после нажатия сочетания набираем искомый файл по первым символам наименования)

Ctrl+Enter, Alt+Enter

Alt+A - копирует полный путь к объекту текущей панели в командную строку

Alt+Shift+? – вызов окна расширенного поиска

Alt+O – показать содержимое каталога под указателем в соседней панели

Представленных «горячих клавиш» достаточно для выполнения данной лабораторной работы. Подробнее вы можете узнать самостоятельно.