Лабораторная работа №2

Знакомство с традиционными средствами разработки в ОС Linux

Цели работы

- Знакомство с традиционными средствами разработки для Linux
  - Пакет компиляторов gcc
  - Система сборки GNU make
- Сборка программных пакетов из исходного кода
- Знакомство с системой управления пакетов на примере дистрибутива Ubuntu

Необходимые навыки

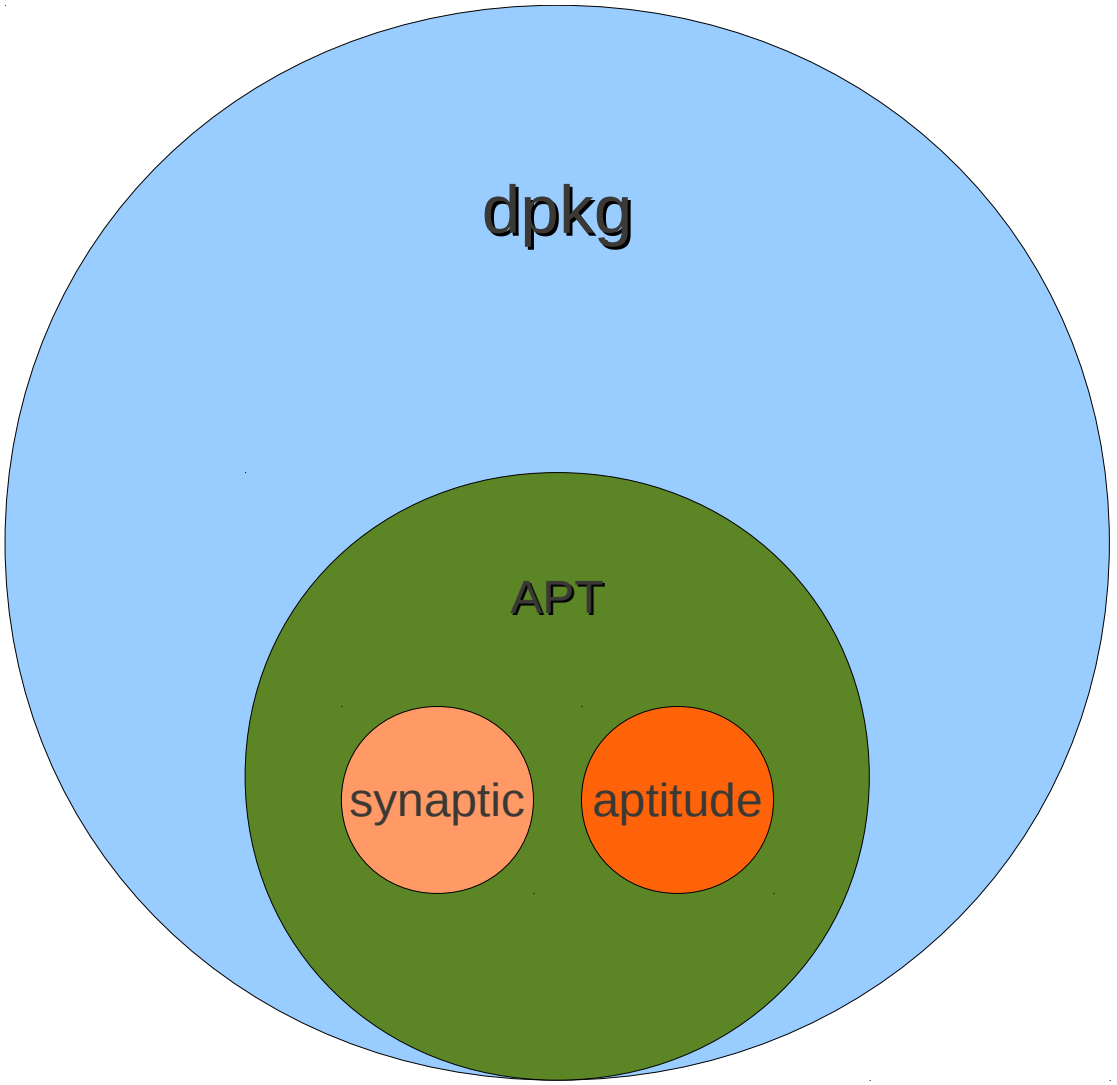
- Базовое знание языков программирования C/C++
- Желательно базовое знакомство с основными служебными программами Linux (`ls`, `rm`, `mkdir` и т.п.)

Необходимые программные и аппаратные средства

- ПК под ОС Linux (возможно, в виртуальной машине)
- Желательно использовать дистрибутив с развитой системой управления пакетами (dpkg, rpm...)
- Указания в настоящей работе были проверены на дистрибутиве Ubuntu

Установка необходимых программных пакетов

Система управления пакетами dpkg



Установка пакетов

- dpkg
  - `apt-get install <название пакета>`
  - `aptitude install <название пакета>`
- rpm
  - `yum install <название пакета>`

- установите пакеты make, gcc

Команды следует исполнять с правами пользователя root.
Например, `sudo apt-get install make`

Пакет компиляторов GNU Compiler Collection

Компиляция HelloWorld

Пример в каталоге 01

- Простейший случай:
  - `gcc hello.c`
  - по устоявшейся традиции вывод генерируется в файл `a.out`
  - Запускаем полученный исполняемый файл: `./a.out`
- Задаём название итогового файла:
  - `gcc -o hello hello.c`
  - Запускаем: `./hello`

Сборка многомодульного приложения

пример в каталоге 02

- Компилируем (и ассемблируем) модули
  - `gcc -c -o hello_main.o hello_main.c`
  - `gcc -c -o hello_util.o hello_util.c`
- Линкуем модули в один исполняемый файл:
  - `gcc -o hello hello_main.o hello_util.o`

ключ `-c` указывает gcc не выполнять линковку

Линковка с внешними библиотеками

пример в каталоге 03

- Устанавливаем библиотеку ncurses
  - `apt-get install libncurses-dev`
- Компилируем и ассемблируем модули (аналогично предыдущему примеру)
- Линкуем модули и библиотеку ncurses в один исполняемый файл
  - `gcc -lncurses -o hello hello_main.o hello_util.o`

Статическая линковка с библиотеками

пример в каталоге 03

- Линкуем уже собранные на предыдущем этапе объектные файлы:
  - `gcc -o hello_static hello_main.o hello_util.o -static -lnucrses`
- Смотрим вывод `ls -lh hello hello_static` Сравниваем размеры файлов.
- **Помните**: использование статической линковки в большинстве случаев неоправдано

Компиляция кода на C++

- gcc компилирует код для C, C++, Objective C, Fortran и т.д.
- Язык “угадывается” по расширению файла, т.е. `gcc -c myfile.cpp` сработает.

НО:

- в таком варианте не подключается стандартная библиотека C++, поэтому –
- Используем обёртку вокруг gcc – g++
- Аргументы такие же, как у gcc

Часто используемые ключи gcc

- Добавление отладочной информации
  - Собираем пример из каталога 01: `gcc -g -o hello hello.c`
  - Смотрим ассемблерный листинг с привязкой к строкам кода: `objdump -S hello | less`
- Включение всех предупреждений (warnings):
  - Для примера из каталога 01: `gcc -Wall -o hello hello.c`

Часто используемые ключи gcc

- Указание каталогов, в которых будут искаться заголовочные файлы: `-I`
- Указание катологов, в которых будут искаться библиотеки: `-L`
- Определение макроконстант: `-D`
  - пример: `gcc -DDEBUG -o hello hello.c`
  - в коде можно использовать условный блок

```
#ifdef DEBUG
```

Часто используемые ключи gcc

- Оптимизация кода
  - `-O0, -O1, -O2, -O3` : различные уровни оптимизации. Чем выше уровень, тем больше включено оптимизаций
  - `-Os` : минимизация размера итогового двоичного файла
  - `-mtune=cpu, -march=cpu` : оптимизация, использующая особенности архитектуры целевой платформы
    - пример `-mtune=core2`

Система сборки GNU make

Запуск сборки с использованием GNU make

задание в каталоге 04/sample

- выполните команду make
  - make ищет файлы с названием `makefile, Makefile...`
  - будет собрана первая по порядку описания в файле цель (target)
- выполните команду `make -f Makefile.3`
  - в качестве makefile будет использоваться файл `Makefile.3`
- выполните команду `make -f Makefile.3 clean`
  - будет собрана цель `clean`
- выполните команду `make -f Makefile.3 -j 2`
  - параллельная сборка: make будет запускать одновременно по две команды

Пишем Makefile для простого проекта

задание в каталоге 04

- Описание задания:

По примерам make-файлов в подкаталоге `sample` написать make-файлы для проекта “ncurses Snake” в подкаталоге `task`

Пишем Makefile – этап 0

пример Makefile.0

- Формат правила:

```
target: prerequisite1 … prerequisiteN
        recipe
        ...
```

  - target – какую цель (какой файл) мы хотим получить на данном этапе
  - prerequisites – какие цели (какие другие файлы) нам для этого требуются
  - recipe – какие команды надо выполнить, чтоб получить target

Пишем Makefile – этап I

пример Makefile.1

- добавляем цели для сборки всего проекта и для удаления результатов предыдущей сборки, традиционно называющиеся all и clean
- цели `all` и `clean` объявляются как `.PHONY`: им не соответствуют реальные файлы. Это избавит нас от неприятностей, если в директории сборки окажутся файлы с такими названиями
- В командах используем особые переменные:
  - `@` – разворачивается в название цели
  - `<` – разворачивается в первый элемент списка реквизитов
  - `^` - разворачивается в список реквизитов

Пишем Makefile – этап II

пример Makefile.2

- параметризуем команды сборки при помощи переменных
- запустите `make -f Makefile.2 CFLAGS='-O0 -g'` и убедитесь, что компиляция будет запускаться именно с этими ключами

Пишем Makefile – этап III

пример Makefile.3

- Используем implicit rules:
  - не указываем “recipe” для цели
  - используется стандартный “recipe” заданный для данного типа файлов
  - recipe для цели `n.o`, если существует файл `n.c`:

```
$(CC) $(CPPFLAGS) $(CFLAGS) -c -o n.o n.c
```

  - recipe для цели n.o, если существует файл n.cc/n.cpp/n.C:

```
$(CXX) $(CPPFLAGS) $(CXXFLAGS) -c -o n.o n.cpp
```

Сборка программных пакетов из исходного кода

Собираем интерпретатор командной строки bash

- Скачиваем с сайта GNU исходные коды последней версии bash:
  - `wget http://ftp.gnu.org/gnu/bash/bash-4.1.tar.gz`
- Распаковываем архив приложением tar:
  - tar xzf bash-4.1.tar.gz
- В полученном каталоге запускаем конфигурационный скрипт:
  - ./configure

  можно запустить `configure --help` *для того, чтоб узнать, какие агрументы можно передать configure*
- Запускаем сборку командой `make all`
- *Опционально*: устанавливаем пакет командой `make install`

Truly open source

- Скачать исходники вашего любимого приложения, изменить и собрать их очень просто!
- Скачиваем исходники при помощи менеджера пакетов:
  - `apt-get source <название пакета>`
- Устанавливаем пакеты, необходимые для сборки вашего пакета:
  - `sudo apt-get build-dep <название пакета>`
- Конфигурируем, собираем и устанавливаем пакет
- Задание: самостоятельно скачайте и соберите приложение grep

Для дополнительной информации

- man pages (руководство): просмотр вызывается командой

```
man PAGE
```

где PAGE – название страницы руководства. Например:

```
man gcc
```

- Документация по GCC (http://gcc.gnu.org/onlinedocs/)
- Документация по GNU make (http://www.gnu.org/software/make/manual/make.html)

Контакт для дополнительных вопросов:

sergeyle@gmail.com