Лабораторная работа №1

Программирование в UNIX

Цель: *Ознакомиться с работой системных компиляторов gcc и g++; ознакомиться с элементарными системными вызовами getuid(), getgid(), getpid() и getppid(); изучить работу с параметрами функции main() в языке C.*

1.Теоретический блок

Написание, компиляция и запуск программ на языке C/C++ в ОС UNIX

В UNIX-подобных операционных системах стандартным средством компиляции программ на языках *C* и *C++* являются компиляторы *gcc* и *g++* соответственно. Рассмотрим их работу на примере *gcc*.

Написание исходного кода программ можно производить в обычном текстовом редакторе. Для того чтобы *gcc* нормально работал, необходимо, чтобы исходные файлы, содержащие текст программы, имели имена, заканчивающиеся на *.c* (*.cpp* для *C++*).

В простейшем случае откомпилировать программу можно, запуская компилятор командой

gcc имя_исходного_файла

Если программа была написана без ошибок, то компилятор создаст исполняемый файл с именем *a.out*. Изменить имя создаваемого исполняемого файла можно, задав его с помощью опции *-o*.

gcc имя_исходного_файла -o имя_исполняемого_файла

Компилятор *gcc* имеет несколько сотен возможных опций. Получить информацию о них вы можете в UNIX Manual.

Запустить программу на исполнение можно, набрав имя исполняемого файла и нажав клавишу <Enter>. Если исполняемый файл находится в текущей директории, то для его запуска достаточно выполнить команду

./ имя_исполняемого_файла

Утилита make/gmake

Когда проект состоит из множества файлов, то любое изменение в одном из них неизбежно влечет за собой перекомпиляцию всех остальных, облегчить эту задачу способна утилита *make* (в некоторых системах она называется *gmake*). Этой утилите нужно передать простой текстовый файл под названием *Makefile*, который содержит информацию о правилах сборки и зависимостях. Правила записываются в следующем виде:

<цель>: <зависимости>

<команда>

<команда>

...

При этом каждая строка с командой должна начинаться с табуляции. Первая цель в *Makefile* выполняется по умолчанию при запуске *make* без аргументов. Ее принято называть *all*, что эквивалентно команде "*make all*". Пример *Makefile*:

```
all: you_prog
you_prog: you_prog.o foo.o boo.o
 gcc you_prog.o foo.o boo.o -o you_prog
you_prog.o: you_prog.c you_prog.h
foo.o: foo.c foo.h
boo.o: boo.c boo.h
clean:
 rm -f *.o you_prog
```

Цель «*clean*» предназначена для удаления всех сгенерированных объектных файлов и программ, чтобы *make* могла создать их заново. Чтобы собрать проект достаточно в командной строке набрать:

make

В *man* об утилите *make* можно узнать много других интересных подробностей.

Утилиты automake/autoconf

Но есть еще один более простой способ создания Make-файлов, с помощью стандартных утилит *automake* и *autoconf*. Сначала нужно подготовить файл *Makefile.am*, например:

bin_PROGRAMS = you_prog

you_prog_SOURCES = you_prog.c foo.c boo.c

AUTOMAKE_OPTIONS = foreign

Последняя опция указывает на то, что в проект не будут включаться файлы стандартной документации: NEWS, README, AUTHORS и ChangeLog. Согласно стандарту их присутствие в GNU-пакете обязательно. Теперь нужно создать файл *configure.in*. Это можно сделать с помощью утилиты *autoscan*. *Autoscan* выполняет анализ дерева исходных текстов, корень которого указан в командной строке или совпадает с текущим каталогом, и создает файл *configure.scan*. Нужно просмотреть *configure.scan*, внести необходимые коррективы и затем переименовать в *configure.in*. И последним этапом следует запустить утилиты в следующем порядке:

aclocal

autoconf

automake -a -c

В результате в текущей директории появятся скрипты *configure*, *Makefile.in* и файлы документации. Чтобы собрать проект достаточно ввести следующие команды:

./configure

make

Утилиты *autoconf* и *automake* входят в серию *Autotools*.

Параметры функции main() в языке С. Переменные среды и аргументы командной строки

У функции *main()* в языке программирования *С* существует три параметра, которые могут быть переданы ей операционной системой. Полный прототип функции *main()* выглядит следующим образом:

*int main(int argc, char *argv[], char *envp[]);*

Первые два параметра при запуске программы на исполнение командной строкой позволяют узнать полное содержание командной строки. Вся командная строка рассматривается как набор слов, разделенных пробелами. Через параметр *argc* передается количество слов в командной строке, которой была запущена программа. Параметр *argv* является массивом указателей на отдельные слова. Так, например, если программа была запущена командой

a.out 12 abed

то значение параметра *argc* будет равно 3, *argv[0]* будет указывать на имя программы — первое слово — *"a.out"*, *argv[l]* — на слово *"12"*, *argv[2]* — на слово *"abed"*. Так как имя программы всегда присутствует на первом месте в командной строке, то *argc* всегда больше 0, а *argv[0]* всегда указывает на имя запущенной программы.

Анализируя в программе содержимое командной строки, мы можем предусмотреть ее различное поведение в зависимости от слов, следующих за именем программы. Таким образом, не внося изменений в текст программы, мы можем заставить ее работать по-разному от запуска к запуску. Например, компилятор *gcc*, вызванный командой *gec 1.c* будет генерировать исполняемый файл с именем *a. out*, а при вызове командой *gec 1.c -o 1.exe* — файл с именем *1.exe*.

Третий параметр — *envp* — является массивом указателей на параметры окружающей среды процесса. Начальные параметры окружающей среды процесса задаются в специальных конфигурационных файлах для каждого пользователя и устанавливаются при входе пользователя в систему. В дальнейшем они могут быть изменены с помощью специальных команд операционной системы UNIX. Каждый параметр имеет вид: *переменная=строка*. Такие переменные используются для изменения долгосрочного поведения процессов, в отличие от аргументов командной строки. Например, задание параметра *TERM=vtl100* может говорить процессам, осуществляющим вывод на экран дисплея, что работать им придется с терминалом *vtl00*. Меняя значение переменной среды *TERM*, например на *TERM=console*, мы сообщаем таким процессам, что они должны изменить свое поведение и осуществлять вывод для системной консоли.

Размер массива аргументов командной строки в функции *main()* мы получали в качестве ее параметра. Так как для массива ссылок на параметры окружающей среды такого параметра нет, то его размер определяется другим способом. Последний элемент этого массива содержит указатель *NULL*.

Системные вызовы getuid и getgid

Каждый пользователь в UNIX-подобной операционной системе имеет свой собственный идентификатор (PID) и идентификатор группы, к которой он относится (GID). Узнать идентификатор пользователя, запустившего программу на исполнение, и идентификатор группы, к которой он относится, можно с помощью системных вызовов *getuid()* и *getgid()*, применив их внутри этой программы.

Прототипы системных вызовов

```
#include <sys/types.h>
#include <unis.td.h>
uid_t getuid(void);
gid_t getgid(void);
```

Системный вызов *getuid* возвращает идентификатор пользователя для текущего процесса. Системный вызов *getgid* возвращает идентификатор группы пользователя для текущего процесса.

Типы данных *uid_t* и *gid_t* являются синонимами для одного из целочисленных типов языка *C*.

Системные вызовы getppid() и getpid()

Каждый процесс, работающий в UNIX-подобной операционной системе, имеет свой собственный идентификатор (PID) и идентификатор родительского процесса (PPID). Значение идентификатора текущего процесса может быть получено с помощью системного вызова *getpid()*, а значение идентификатора родительского процесса для текущего процесса — с помощью системного вызова *getppid()*. Прототипы этих системных вызовов и соответствующие типы данных описаны в системных файлах *<sys/types.h>* и *<unistd.h>*. Системные вызовы не имеют параметров и возвращают идентификатор текущего процесса и идентификатор родительского процесса соответственно.

Прототипы системных вызовов

```
#include <sys/types.h>
#include <unistd.h>
pid_t getpid(void);
pid_t getppid(void);
```

Системный вызов *getpid* возвращает идентификатор текущего процесса.

Системный вызов *getppid* возвращает идентификатор процесса-родителя для текущего процесса.

Тип данных *pid_t* является синонимом для одного из целочисленных типов языка *C*.

2. Практические задания

Задание 1. *Компиляция программы. Работа с системными вызовами*

Напишите, откомпилируйте и запустите программу выводящую идентификатор пользователя, идентификатор группы, идентификатор процесса и идентификатор родительского процесса. Запустите программу несколько раз, посмотрите, какие идентификаторы меняются и как, объясните это явление.

Задание 2. *Параметры функции main(). Работа с утилитой make*

Напишите программу, распечатывающую значения аргументов командной строки и параметров окружающей среды для текущего процесса. Создайте makefile для компиляции этой программы. Запустите программу на выполнение.

Литература

1. В.Е. Карпов, К.А. Коньков Основы операционных систем. Курс лекций. Учебное пособие. Издание второе, дополненное и исправленное. М.: ООО «ИНТУИТ.ру» 2005
2. И. Скляров Стандартные утилиты для UNIX-программиста // http://www.sklyaroff.ru/x10.php