4. ОКРУЖЕНИЕ

4.1. Введение в окружение

Окружение (environment) или среда - это набор пар ПЕРЕМЕННАЯ=ЗНАЧЕНИЕ, доступный каждому пользовательскому процессу. Иными словами, окружение - это набор переменных окружения. Если вы используете оболочку, отличную от bash, то не все примеры этой главы могут быть воспроизведены.

Для того, чтобы посмотреть окружение, просто введите команду env без аргументов. В зависимости от конфигурации системы, вывод env может занять несколько экранов, поэтому лучше сделать так:

```
$ env > myenv
$
```

Или так:

```
$ env | more
```

Или так:

```
$ env | less
$
```

Переменные окружения могут формироваться как из заглавных, так и из строчных символов, однако исторически сложилось именовать их в верхнем регистре. Мы также не будем отступать от этого неписанного правила.

Про полезность окружения можно говорить долго, но основное его назначение - заставить одни и те же программы работать у разных пользователей по-разному. Приятно, например, когда программа "угадывает" имя пользователя или домашний каталог пользователя. Чаще всего такая информация "добывается" из переменных окружения USER и HOME соответственно.

Значение каждой переменной окружения изначально представляет собой строковую константу (строку). Интерпретация значений переменных полностью возлагается на программу. Иными словами, все переменные окружения имеют тип char*, а само окружение имеет тип char**. Чтобы вывести на экран значение какой-нибудь переменной окружения, достаточно набрать echo $ИМЯ_ПЕРЕМЕННОЙ:

```
$ echo $USER
sergio
$ echo $HOME
/home/sergio
$
```

Вообще говоря, при работе с оболочкой bash, запись $ИМЯ_ПЕРЕМЕННОЙ заменяется на само значение переменной, если только эта запись не встречается в кавычках, апострофах или в комментариях. В моем случае, например, запись $HOME заменяется на /home/sergio. То

есть команда mkdir $HOME/mynewdir создаст в моем домашнем каталоге подкаталог mynewdir.

В разных системах и у разных пользователей окружение отличается не только значениями переменных, но и наличием/отсутствием этих переменных. Пользователи, использующие универсальные MUA (Mail User Agent), наподобие Mozilla-mail, Kmail или Sylpheed вряд ли будут иметь в своем окружении (по крайней мере с пользой) переменные MAIL или MAILDIR. А пользователям mutt, pine или elm (с довесками в виде fetchmail/getmail, procmail и проч.) эти переменные жизненно необходимы. Пользователь, не использующий графические оболочки, вряд ли будет иметь в своем окружении переменную QTDIR. Ниже приведены те переменные окружения, которые есть почти у всех пользователей Linux:

- **USER** - имя текущего пользователя
- **HOME** - путь к домашнему каталогу текущего пользователя
- **PATH** - список каталогов, разделенных двоеточиями, в которых производится "поиск" программ
- **PWD** - текущий каталог
- **TERM** - тип терминала
- **SHELL** - текущая командная оболочка

Некоторые переменные окружения имеются не во всех системах, но все-таки требуют упоминания:

- **HOSTNAME** - имя машины
- **QTDIR** - расположение библиотеки QT
- **MAIL** - почтовый ящик
- **LD_LIBRARY_PATH** - место "поиска" дополнительных библиотек (см. предыдущую главу)
- **MANPATH** - место поиска файлов man-страниц (каталоги, разделенные двоеточием)
- **LANG** - язык и кодировка пользователя (иногда LANGUAGE)
- **DISPLAY** - текущий дисплей в X11

Помимо переменных окружения, командные оболочки, такие как bash располагают собственным набором пар ПЕРЕМЕННАЯ=ЗНАЧЕНИЕ. Это переменные оболочки. Набор таких переменных называют окружением (или средой) оболочки. Эти переменные чем-то напоминают локальные (стековые) переменные в языке C. Они недоступны для других программ (в том числе и для env) и используются в основном в сценариях оболочки. Чтобы задать переменную оболочки, достаточно написать в командной строке ПЕРЕМЕННАЯ=ЗНАЧЕНИЕ.

```
$ MYVAR=Hello
$ echo $MYVAR
Hello
$ env | grep MYVAR
$
```

Однако, при желании, можно включить локальную переменную оболочки в основное окружение. Для этого используется команда export:

```
$ export MYVAR
$ env | grep MYVAR
MYVAR=Hello
$
```

Можно сделать сразу так:

```
$ export MYNEWVAR=Goodbye
$ echo $MYNEWVAR
Goodbye
$ env | grep MYNEWVAR
MYNEWVAR=Goodbye
$
```

Прежде, чем продолжать дальше, попробуйте поиграться с переменными окружения, чтобы лучше все понять. Выясните экспериментальным путем, чувствительны ли к регистру символов переменные окружения; можно ли использовать в качестве значений переменных окружения строки, содержащие пробелы; если можно, то как?

Теперь разберемся с тем, откуда берется окружение. Любая запущенная и работающая в Linux программа - это процесс. Запуская дважды одну и ту же программу, вы получаете два процесса. У каждого процесса (кроме init) есть свой процесс-родитель. Когда вы набираете в командной строке vim, в системе появляется новый процесс, соотвествующий текстовому редактору vim; родительским процессом здесь будет оболочка (bash, например). Для самой оболочки новый процесс будет дочерним. Мы будем подробно изучать процессы в последующих главах книги. Сейчас же важно одно: **новый процесс получает копию родительского окружения.** Из этого правила существует несколько исключений, но мы пока об этом говорить не будем. Важно то, что у каждого процесса своя **независимая копия** окружения, с которой процесс может делать все что угодно. Если процесс завершается, то копия теряется; если процесс породил другой, дочерний процесс, то этот новый процесс получает копию окружения своего родителя. Мы еще неоднократно столкнемся с окружением при изучении многозадачности.

4.2. Массив environ

Теперь, когда мы разобрались, что такое окружение, самое время написать программу для взаимодействия с окружением. Чтобы показать, как это все работает, сначала изобретем велосипед.

В заголовочном файле unistd.h объявлен внешний двумерный массив environ:

```
extern char ** environ;
```

В этом массиве хранится копия окружения процесса. Массив не константный, но я не рекомендую вам изменять его - это опасно (для программы) и является плохим стилем программирования. Для изменения environ есть специальные механизмы, которые мы рассмотрим чуть позже. Уверен, что настоящие будущие хакеры прочитают это и сделают с точностью до "наоборот".

А читать environ нам никто не запрещал. Напишем одноименную программу (environ), которой в качестве аргумента передается имя переменной. Программа будет проверять, существует ли эта переменная в окружении; и если существует, то каково ее значение. Как мы позже узнаем, это можно было бы сделать значительно проще. Но я предупредил: мы изобретаем велосипед. Вот эта программа:

```
/* environ.c */
#include <stdio.h>
#include <stdlib.h>
```

```c
#include <unistd.h>
#include <string.h>

extern char ** environ; /* Environment itself */

int main (int argc, char ** argv)
{
        int i;
        if (argc < 2)
        {
                fprintf (stderr, "environ: Too few arguments\n");
                fprintf (stderr, "Usage: environ <variable>\n");
                exit (1);
        }

        for (i = 0; environ[i] != NULL; i++)
        {
                if (!strncmp (environ[i], argv[1], strlen (argv[1])))
                {
                        printf ("'%s' found\n", environ[i]);
                        exit (0);
                }
        }
        printf ("'%s' not found\n", argv[1]);
        exit (0);
}
```

А вот Makefile для этой программы (если нужен):

```
# Makefile for environ

environ: environ.c
        gcc -o environ environ.c

clean:
        rm -f environ
```

Проверяем:

```
$ make
gcc -o environ environ.c
$ ./environ
environ: Too few arguments
Usage: environ <variable>
$ ./environ USER
'USER=nn' found
$ ./environ ABRAKADABRA
'ABRAKADABRA' not found
$
```

В приведенном примере мы осуществили простой синтаксический анализ массива environ, так как переменные и значения представлены в нем в обычном виде (ПЕРЕМЕННАЯ=ЗНАЧЕНИЕ). К счастью нам больше не придется осуществлять синтаксический разбор массива environ. О настоящем предназначении этого массива будет рассказано в главе, посвященной многозадачности.

4.3. Чтение окружения: getenv()

В заголовочном файле stdlib.h объявлена функция getenv , которая доказывает, что в предыдущем примере мы изобрели велосипед. Ниже приведен адаптированный прототип этой функции.

```
char * getenv (const char * name);
```

Функция эта работает очень просто: если в качестве аргумента указано имя существующей переменной окружения, то функция возвращает указатель на строку, содержащую значение этой переменной; если переменная отсутствует, возвращается NULL.

Как видим, функция getenv() позволяет не осуществлять синтаксический разбор environ. Напишем новую программу, которая делает то же, что и предыдущая, только более простым способом. Назовем ее getenv по имени функции - виновника торжества.

```
/* getenv.c */
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char ** argv)
{
        if (argc < 2)
        {
                fprintf (stderr, "getenv: Too few arguments\n");
                fprintf (stderr, "Usage: getenv <variable>\n");
                exit (1);
        }
        char * var = getenv (argv[1]);
        if (var == NULL)
        {
                printf ("'%s' not found\n", argv[1]);
                exit (0);
        }
        printf ("'%s=%s' found\n", argv[1], var);
        exit (0);
}
```

4.4. Запись окружения: setenv()

Пришла пора модифицировать окружение! Еще раз напоминаю: каждый процесс получает не доступ к окружению, а **копию** окружения родительского процесса (в нашем случае это командная оболочка). Чтобы добавить в окружение новую переменную или изменить существующую, используется функция setenv, объявленная в файле stdlib.h. Ниже приведен прототип этой функции.

```
int setenv (const char * name, const char * value, int overwrite);
```

Функция setenv() устанавливает значение (второй аргумент, value) для переменной окружения (первый аргумент, name). Третий аргумент - это флаг перезаписи. При ненулевом флаге уже существующая переменная перезаписывается, при нулевом флаге переменная, если уже существует, - не перезаписывается. В случае успешного завершения setenv() возвращает нуль (даже если существующая переменная не перезаписалась при overwrite==0).

Если в окружении нет места для новой переменной, то setenv() возвращает -1.

Наша новая программа setenv читает из командной строки два аргумента: имя переменной и значение этой переменной. Если переменная не может быть установлена, выводится ошибка, если ошибки не произошло, выводится результат в формате ПЕРЕМЕННАЯ=ЗНАЧЕНИЕ. Вот эта программа:

```
/* setenv.c */
#include <stdio.h>
#include <stdlib.h>

#define FL_OVWR         0       /* Overwrite flag. You may change it. */

int main (int argc, char ** argv)
{
        if (argc < 3)
        {
                fprintf (stderr, "setenv: Too few arguments\n");
                fprintf (stderr,
                        "Usage: setenv <variable> <value>\n");
                exit (1);
        }
        if (setenv (argv[1], argv[2], FL_OVWR) != 0)
        {
                fprintf (stderr, "setenv: Cannot set '%s'\n", argv[1]);
                exit (1);
        }

        printf ("%s=%s\n", argv[1], getenv (argv[1]));
        exit (0);
}
```

Изменяя константу FL_OVWR можно несколько изменить поведение программы по отношению к существующим переменным окружения. Еще раз напоминаю: у каждого процесса своя копия окружения, которая уничтожается при завершении процесса. Экспериментируйте!

4.5. Сырая модификация окружения: putenv()

Функция putenv(), объявленная в заголовочном файле stdlib.h вызывается с единственным аргументом - строкой формата ПЕРЕМЕННАЯ=ЗНАЧЕНИЕ или просто ПЕРЕМЕННАЯ. Обычно такие преформатированные строки называют запросами. Если переменная отсутствует, то в окружение добавляется новая запись. Если переменная уже существует, то текущее значение перезаписывается. Если в качестве аргумента фигурирует просто имя переменной, то переменная удаляется из окружения. В случае удачного завершения, putenv() возвращает нуль и -1 - в случае ошибки.

У функции putenv() есть одна особенность: указатель на строку, переданный в качестве аргумента, становится частью окружения. Если в дальнейшем строка будет изменена, будет изменено и окружение. Это очень важный момент, о котором не следует забывать. Ниже приведен прототип функции putenv:

```
int putenv (char * str);
```

Теперь напишем программу, использующую putenv(). Вот она:

```
/* putenv.c */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define QUERY_MAX_SIZE          32
char * query_str;

void print_evar (const char * var)
{
        char * tmp = getenv (var);
        if (tmp == NULL)
        {
                printf ("%s is not set\n", var);
                return;
        }
        printf ("%s=%s\n", var, tmp);
}

int main (void)
{
        int ret;
        query_str = (char *) calloc (QUERY_MAX_SIZE, sizeof(char));
        if (query_str == NULL) abort ();

        strncpy (query_str, "FOO=foo_value1", QUERY_MAX_SIZE-1);
        ret = putenv (query_str);
        if (ret != 0) abort ();
        print_evar ("FOO");

        strncpy (query_str, "FOO=foo_value2", QUERY_MAX_SIZE-1);
        print_evar ("FOO");

        strncpy (query_str, "FOO", QUERY_MAX_SIZE-1);
        ret = putenv (query_str);
        if (ret != 0) abort ();
        print_evar ("FOO");

        free (query_str);
        exit (0);
}
```

Программа немного сложнее тех, что приводились ранее, поэтому разберем все по порядку. Сначала создаем для удобства функцию print_evar (PRINT Environment VARiable), которая будет отражать текущее состояние переменной окружения, переданной в качестве аргумента. В функции main() перво-наперво выделяем в куче (heap) память для буфера, в который будут помещаться запросы; заносим адрес буфера в query_str. Теперь формируем строку, и посылаем запрос в функцию putenv(). Здесь нет ничего необычного. Дальше идет демонстрация того, на чем я акцентировал внимание: простое изменение содержимого памяти по адресу, хранящемуся в query_str приводит к изменению окружения; это видно из вывода функции print_evar(). Наконец, вызываем putenv() со строкой, не содержащей символа '=' (равно). Это запрос на удаление переменной из окружения. Функция print_evar() подтверждает это.

Хочу заметить, что putenv() поддерживается не всеми версиями Unix. Если нет крайней необходимости, лучше использовать setenv() для пополнения/модификации окружения.

4.6. Удаление переменной окружения: unsetenv()

Функция unsetenv(), объявленная в stdlib.h, удаляет переменную из окружения. Ниже приведен адаптированный прототип этой функции.

```
int unsetenv (const char * name);
```

Прежде всего хочу обратить внимание на то, что раньше функция unsetenv() ничего не возвращала (void). С выходом версии 2.2.2 библиотеки glibc (январь 2001 года) функция стала возвращать int.

Функция unsetenv() использует в качестве аргумента имя переменной окружения. Возвращаемое значение - нуль при удачном завершении и -1 в случае ошибки. Рассмотрим простую программу, которая удаляет переменную окружения USER (!!!). Для тех, кто испугался, напоминаю еще один раз: каждый процесс работает с собственной копией окружения, никак не связанной с копиями окружения других процессов, за исключением дочерних процессов, которых у нас нет. Ниже приведен исходный код программы, учитывающий исторические изменения прототипа функции unsetenv().

```
/* unsetenv.c */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <gnu/libc-version.h>

#define OLD_LIBC_VERSION        0
#define NEW_LIBC_VERSION        1
#define E_VAR                   "USER"

int libc_cur_version (void)
{
        int ret = strcmp (gnu_get_libc_version (), "2.2.2");
        if (ret < 0) return OLD_LIBC_VERSION;
        return NEW_LIBC_VERSION;
}

int main (void)
{
        int ret;
        char * str;
        if (libc_cur_version () == OLD_LIBC_VERSION)
        {
                unsetenv (E_VAR);
        } else
        {
                ret = unsetenv (E_VAR);
                if (ret != 0)
                {
                        fprintf (stderr, "Cannot unset '%s'\n", E_VAR);
                        exit (1);
                }
        }

        str = getenv (E_VAR);
        if (str == NULL)
        {
                printf ("'%s' has removed from environment\n", E_VAR);
        } else
```

```
        {
                printf ("'%s' hasn't removed\n", E_VAR);
        }
        exit (0);
}
```

В программе показан один из самых варварских способов подстроить код под версию библиотеки. Это сделано исключительно для демонстрации двух вариантов unsetenv(). **Никогда не делайте так в реальных программах**. Намного проще и дешевле (в плане времени), не получая ничего от unsetenv() проверить факт удаления переменной при помощи getenv().

4.7. Очистка окружения: clearenv()

Функция clearenv(), объявленная в заголовочном файле stdlib.h, используется крайне редко для полной очистки окружения. clearenv() поддерживается не всеми версиями Unix. Ниже приведен ее прототип.

```
int clearenv (void);
```

При успешном завершении clearenv() возвращает нуль. В случае ошибки возвращается ненулевое значение.

В большинстве случаев вместо clearenv() можно использовать следующую инструкцию:

```
environ = NULL;
```