# Bài 7
# TƯƠNG TÁC VỚI MÔI TRƯỜNG LINUX

**I. Đối số truyền cho chương trình**

- Đối số của hàm **main()** là nơi nhận đối số dòng lệnh truyền cho chương trình:

```
int main( int argc, char* argv[] )
```

**argc**: số đối số truyền cho chương trình.

**argv**: con trỏ chỉ đến mảng chứa giá trị các đối số.

- Hệ shell của Hệ Điều Hành sẽ tiếp nhận các đối số dòng lệnh và và gửi đến Hệ Điều Hành cùng tên của chương trình. Hệ Điều Hành đưa các giá trị đối số vào mảng **argv**, số tham số được đếm và truyền vào **argc**. Các xử lý sơ bộ đối số như xử lý ký tự đại diện, xử lý khoảng trắng do hệ shell thực hiện. Ví dụ: đặt biến môi trường **LAST_PARAM** trị **'Hello World'**, sau đó dùng biến môi trường này như một tham số:

```
$ export LAST_PARAM='Hello World'
$ echo $LAST_PARAM
Hello World
$ myprog hello byebye $LAST_PARAM
```

- Chương trình **args.c** dưới đây sẽ in tất cả giá trị của đối số mà hàm main nhận được:

**Bài 1: args.c**

```
#include <stdio.h>
int main( int argc, char* argv[] )
{
  int arg;
  for ( arg = 0; arg < argc; arg++ ) {
    if ( argv[arg][0] == '-' )
      printf( "option: %s\n", argv[arg] + 1 );
    else
      printf( "argument %d: %s\n", arg, argv[arg] );
  }
  return 0;
}
```

```
$ ./args –i –lr 'Hello World' –f file.c
argument 0: args
option: i
option: lr
argument 3: HelloWorld
option: f
argument 5: file.c
```

Chú ý tùy chọn **–f** có dữ liệu kết hợp kèm theo là đối số tiếp theo ngay sau nó: **file.c**

- Hàm **getopt()** giúp ta phân tích đối số dòng lệnh tuân theo một khuôn dạng chỉ định.

Hàm **getopt()** nhận **argc** và **argv** làm đối số thứ nhất và thứ hai. Đối số thứ ba là chuỗi xác định nội dung tùy chọn, chuỗi này sẽ thông báo cho **getopt()** biết tùy chọn được định nghĩa như thế nào trong chương trình, có hay không dữ liệu kết hợp với tùy chọn.

Ví dụ: lới gọi hàm: **getopt( argc, argv, "if:lr" );**

sẽ phân tích và xem **–i**, **-f**, **-l** và **–r** là các tùy chọn, tuy nhiên **–f** sẽ có dữ liệu kèm theo (sau **f** có ký tự **:**) là đối số đi liền sau **–f**.

Kết quả trả về của **getopt()** là ký tự tùy chọn kế tiếp được tìm thấy trong mảng **argv**. Vì vậy cần gọi **getopt()** nhiều lần trong vòng lặp để lần lượt lấy ra các tùy chọn mong muốn.

- Hoạt động của **getopt()** như sau:

+ Nếu tùy chọn có kèm theo giá trị dữ liệu, giá trị này sẽ được lấy ra và trỏ đến bởi biến toàn cục **optarg**.
+ **getopt()** trả về mã **-1** nếu không có tùy chọn nào có thể lấy ra được
+ **getopt()** trả về ký tự **?** nếu không nhận dạng được tùy chọn, giá trị không nhận được này sẽ được lấy ra và trỏ đến bởi biến toàn cục **optopt**.
+ Nếu tùy chọn yêu cầu dữ liệu kèm theo mà trong danh sách đối số phân tích không tìm thấy dữ liệu, hàm **getopt()** trả về ký tự **:**.
+ Biến toàn cục **optind** dùng làm chỉ số cho biết vị trí của đối số tiếp theo cần xử lý.

- Chương trình ví dụ **argopt.c** cho thấy cách sử dụng hàm **getopt()**:

**Bài 2: argopt.c**

```
#include <stdio.h>
#include <unistd.h>
int main( int argc, char* argv[] )
{
    int opt;
    while ( ( opt = getopt( argc, argv, "if:lr" )) != -1 )
    {
        switch ( opt )
        {
            case 'i' :
            case 'l' :
            case 'r' :
                printf( "option: %c\n", opt );
                break;
            case 'f' :
                printf( "filename: %s\n", optarg );
                break;
```

```
            case ':' :
                printf( "option needs a value\n" );
                break;
            case '?' :
                printf( "unknown option: %c\n", optopt );
                break;
        }
    }
    for ( ; optind < argc; optind++ )
        printf( "argument: %s\n", argv[optind] );
    return 0;
}
```

```
$ ./argopt –i –lr 'Hello World' –f file.c -q
option: i
option: l
option: r
filename: file.c
./argopt: invalid option –-q
invalid option: q
argument: HelloWorld
```

**II. Biến môi trường (Environment Variables)**

**1. Đọc và thiết lập biến môi trường**

- Linux định nghĩa rất nhiều biến môi trường dùng cho các mục đích khác nhau. Biến môi trường dùng kiểm soát và cung cấp thông tin để các chương trình và script của shell hoạt động, dùng cấu hình môi trường làm việc của người dùng.
- Nội dung một biến môi trường được xem bằng lệnh **echo**. Ví dụ: **echo $HOME**
- Danh sách các biến môi trường trong phiên làm việc hiện hành cùng trị của chúng, được xem bằng lệnh **set**.
- Các chương trình có thể truy xuất và thiết lập biến môi trường dựa vào hàm **getenv()** và **putenv()**.

```
#include <stdlib.h>
char* getenv( const char* name );
int putenv( const char* string );
```

Các biến môi trường thường ở dạng chuỗi: **varname = value**

- Hàm **getenv()** sẽ dò tìm trong môi trường hiện hành chuỗi **varname**, nếu tìm thấy nó sẽ trả về giá trị chuỗi **value** tương ứng với tên **varname**. Nếu tên biến **varname** không tồn tại hoặc giá trị không có, hàm sẽ trả về **null**.
- Hàm **putenv()** tiếp nhận một chuỗi theo mẫu '**varname=value**' và đặt chuỗi này vào danh sách biến môi trường hiện hành. Nếu vùng nhớ chứa các biến môi trường của hệ thống đã hết, hàm sẽ trả về trị **-1**.
- Ví dụ:

**Bài 3: get_setenv.c**

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int main( int argc, char* argv[] )
{
  char* var, * value;
  // Kiểm tra đối số
  if ( argc == 1 || argc > 3 ) {
    fprintf( stderr, "Usage: get_setenv var [value]\n" );
    exit( 1 );
  }
  // Lấy tên biến môi trường rồi tìm trị
  var = argv[1];
  value = getenv( var );
  if ( value )
    pritnf( "Variable %s has value %s\n", var, value );
  else
    pritnf( "Variable %s has no value\n", var );
  // Nếu có đối số thứ ba thì thiết lập trị cho biến môi trường với trị là đối số thứ ba
  if ( argc == 3 ) {
    char* string;
    value = argv[2];
    string = malloc( strlen( var ) + strlen( value ) + 2 );
    if ( !string ) {
      fprintf( stderr, "Out of memory\n" );
      exit( 1 );
    }
    strcpy( string, var );
    strcat( string, "=" );
    strcat( string, value );
    printf( "Calling putenv() with: %s\n", string );
    if ( putenv( string ) != 0 ) {
      fprintf( stderr, "putenv() failed\n" );
      free( string );
      exit( 1 );
    }
    // Lấy giá trị mới của biến môi trường
    value = getenv( var );
```

```
    if ( value )
      pritnf( "New value of %s is %s\n", var, value );
    else
      pritnf( "New value of %s is null?\n", var );
  }
  return 0;
}
```

Test chương trình:

```
$ get_setenv HOME
Variable HOME has value /home/dsl
$ get_setenv BOOK
Variable BOOK has no value
$ get_setenv  BOOK Linux_Programming
Variable BOOK has no value
Calling putenv with: BOOK = Linux_Programming
New value of BOOK is Linux_Programming
$get_setenv BOOK
Variable BOOK has no value
```

**2. Sử dụng biến môi trường**

- Biến môi trường do ta tạo trong chương trình trên chỉ có giá trị cục bộ đối với chương trình. Muốn biến môi trường trở nên toàn cục trở nên toàn cục và thấy được bởi chương trình, phải dùng lệnh export, ví dụ:

```
$ MYVAR=Linux
$ echo $MYVAR
Linux
$ get_setenv MYVAR
Variable MYVAR has no value
$ export MYVAR=Linux
$ get_setenv MYVAR
Variable MYVAR has value Linux
```

**3. Biến environ**

- Danh sách tất cả các biến môi trường mà chương trình có thể đọc được chứa trong biến **environ**, là mảng các chuỗi theo dạng **varname=value**:

```
#include <stdlib.h>
extern char** environ;
```

Ví dụ **showenv.c** cho thấy cách đọc nội dung biến **environ**:

**Bài 4: showenv.c**

```
#include <stdlib.h>
#include <stdio.h>
extern char** environ;
int main()
{
  char** env = environ;
  while ( *env ) {
    pritnf( "%s\n", *env );
    env++;
  }
  return 0;
}
```

Kết quả khi chạy chương trình:

```
dsl@box:~$ ./showenv
HZ=100
SHELL=/bin/bash
TERM=rxvt
WINDOWID=12582914
USER=dsl
LS_COLORS=no=00:fi=00:di=01;34:ln=01;36:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31;01:ex=01;32:*.tar=01;31:*.tgz
=01;31:*.arj=01;31:*.taz=01;31:*.lzh=01;31:*.zip=01;31:*.z=01;31:*.Z=01;31:*.gz=01;31:*.bz2=01;31:*.deb=01;31:*.rpm=01;31:*.jar=0
1;31:*.jpg=01;35:*.jpeg=01;35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.t
if=01;35:*.tiff=01;35:*.png=01;35:*.mpg=01;35:*.mpeg=01;35:*.avi=01;35:*.fli=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;35:
*.ogg=01;35:*.mp3=01;35:*.wav=01;35:
MAIL=/var/mail/dsl
PATH=/sbin:/bin:/usr/sbin:/usr/bin:/usr/X11R6/bin:/usr/local/sbin:/usr/local/bin:/usr/games:/opt/bin:.:.
PWD=/home/dsl
LANG=C
COLORFGBG=15;0
IRCNICK=DSL
SHLVL=5
HOME=/home/dsl
LANGUAGE=us
LOGNAME=dsl
DISPLAY=:0.0
COLORTERM=rxvt-xpm
XAUTHORITY=/home/dsl/.Xauthority
_=./showenv
dsl@box:~$
```

**III. Thông tin về người dùng**

- Người dùng thường triệu gọi chương trình (lệnh) từ dấu nhắc của shell, ta sẽ tìm hiểu thông tin người dùng đang sử dụng chương trình.
- Khi người dùng đăng nhập, hệ thống sẽ kiểm tra username và password. Nếu hợp lệ, hệ thống cho phép tiếp cận dấu nhắc của shell và cấp cho mỗi người dùng một số định danh UID dùng trong hệ thống. Chương trình do người dùng gọi sẽ mang số UID cho biết người dùng đã gọi nó. Linux cung cấp các hàm sau để lấy định danh UID cũng như thông tin người dùng đã đăng nhập triệu gọi chương trình:

```
#include <unistd.h>
#include <sys/types.h>

uid_t getuid( void );
char* getlogin();
```

**getuid()** trả về UID của người dùng đang chạy chương trình, là một số nguyên.
**getlogin()** trả về chuỗi thông tin về người dùng tương ứng với số UID lấy được.

- Thông tin người dùng chứa trong file /etc/passwd và ánh xạ trong file /etc/shadow (xem bởi root). Có thể lấy bằng cách dùng các hàm sau:

```
#include <sys/types.h>
#include <pwd.h>
struct passwd* getpwuid( uid_t uid );
struct passwd* getpwname( const char* name );
```

Cấu trúc **passwd** trả về cung cấp thông tin người dùng:

```
struct passwd {
  char* pw_name;              /* Tên người dùng */
  uid_t pw_uid;               /* Định danh người dùng UID */
  gid_t pw_gid;               /* Định danh nhóm GUID */
  char* pw_dir;               /* Đường dẫn thư mục chủ */
  char* pw_shell;             /* Shell thực thi khi người dùng đăng nhập */
};
```

- Ví dụ **user_info.c** hiển thị thông tin về người dùng đang sử dụng chương trình:

**Bài 5: user_info.c**

```
#include <unistd.h>
#include <sys/types.h>
#include <stdio.h>
#include <pwd.h>
int main()
{
  uid_t uid;
  gid_t gid;
  struct passwd* pw;

  uid = getuid();
  gid = getgid();
  printf( "User is %s\n", getlogin() );
  printf( "User IDs: uid=%d, gid=%d\n", uid, gid );

  pw = getpwuid( uid );
  printf( "UID passwd entry:\n name=%s, uid=%d, gid=%d, home=%s, shell=%s\n",
          pw->pw_name, pw->pw_uid, pw->pw_gid, pw->pw_dir, pw->pw_shell );

  pw = getpwnam( "root" );
  printf( "root passwd entry:\n" );
  printf( "name=%s, uid=%d, gid=%d, home=%s, shell=%s\n",
          pw->pw_name, pw->pw_uid, pw->pw_gid, pw->pw_dir, pw->pw_shell );
  return 0;
}
```

Biên dịch và chạy chương trình với kết quả kết xuất như sau:

```
sh-2.05b$ ./user_info
User is: (null)
User IDs: uid = 1002, gid = 100
UID passwd entry:
name = user1, uid = 1002, gid = 100, home = /home/user1, shell =
root password entry:
name = root, uid = 0, gid = 0, home = /root, shell = /bin/bash
sh-2.05b$
```

- Để lấy về danh sách tất cả người dùng, sử dụng hàm **getpwent()**.