Lập trình Hệ thống

Unix Programming

Part 2: C Programming

Nguyễn Quốc Tuấn

Network and Communication System Department
Faculty of Electronics and Communications
UNIVERSITY OF ENGINEERING AND TECHNOLOGY

C Programming

❑ THIẾT LẬP MÔI TRƯỜNG C

- Ngôn ngữ C đã có sẵn trong core của Unix hay Linux (Ubuntu). Các nhà thiết kế hệ điều hành Unix mong muốn được viết trên C
- Ngôn ngữ C hiện nay là ngôn ngữ chuyên nghiệp được đưa vào giảng dạy những năm 1990. Cú pháp C khác dễ dàng để giúp người lập trình hiểu được các ngôn ngữ khác như C++, Java hay C#

- Thiết lập môi trường lập trình ta cần cài đặt các gói phần mềm gồm:
  + Các gói thư viện
  + Các gói chứa các file header cho các lời gọi thư viện
  + Các gói chứa các tiện ích (biên dịch, đóng gói, gỡ rối,…)

- Chương trình ngôn ngữ C trên Unix chuẩn (ISO C)
  + Với Linux hay Ubuntu cho phép ngôn ngữ C hay C++
  + C trở thành ngôn ngữ mặc định có sẵn (build-in) HDH Unix

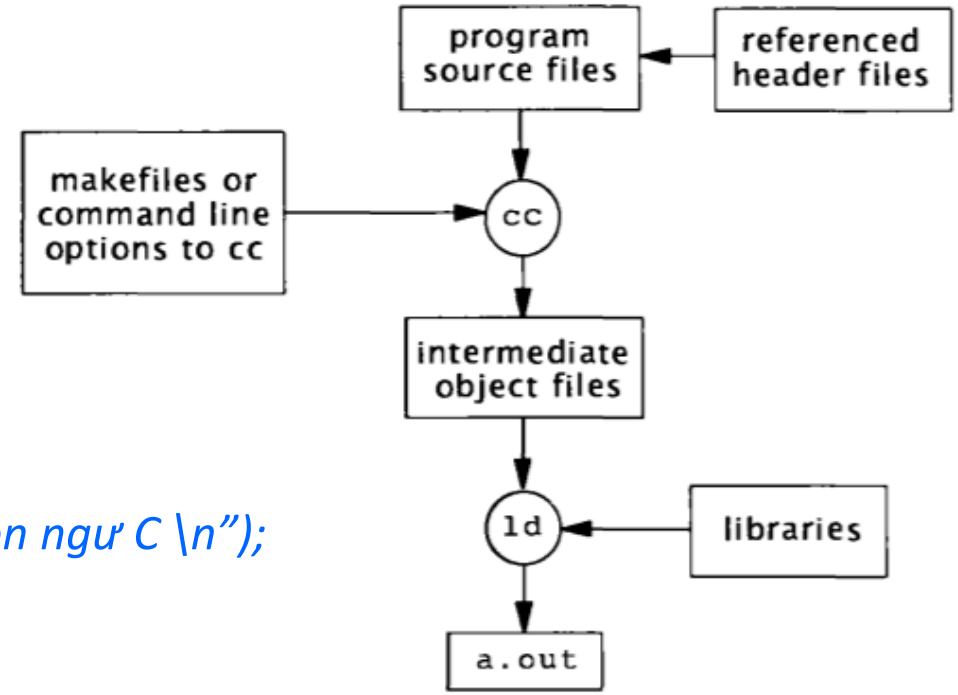
C Programming

❑ THIẾT LẬP MÔI TRƯỜNG C

- **Thủ tục chương trình ngôn ngữ C**

- Dùng tiến trình ***vi*** để soạn thảo
- Ví dụ: *$ vi hello.c*

```
#include <stdio.h>
Int main (void)
{
    printf("Hello, chung ta dang hoc ngon ngư C \n");
    return 0;
}
```



- Dùng lệnh *gcc/cc* để dịch từ file.c ra các dạng file bin

$ cc –o <file dích> <file nguồn> Tạo ra file vận hành được

$ cc –c <file đích> <file nguồn> Tạo ra file object

$ ld –o <file đích> <file object> <library>

❑ THIẾT LẬP MÔI TRƯỜNG C

- **Thủ tục chương trình ngôn ngữ C**

- Sau khi compile *$cc –o hello hello.c /*Tạo ra file hello dạng bin có thể chạy được*/*

  Khi ta dùng lệnh *ls –l* ta sẽ thấy

  -rwxrwxr-x 1 qtuan qtuan 16688 Oct 10 14:01 hello

  Để vận hành ta sử dụng

  ./hello

  Hello, chung ta dang hoc ngon ngu C

- *# include <stdio.h>* file thư viện header cho phép chương trình sử dụng lệnh print(.)
- *main (void)* chỉ điểm băt đầu của chương trình, và được dung để phân biệt với các hàm trong chương trình.

C Programming

❑ THIẾT LẬP MÔI TRƯỜ

- Một số tùy chọn khi dịch (compiler)

Tuỳ chọn	Ý nghĩa
-o FILE	Chỉ định tên file đầu ra; không cần thiết khi biên dịch sang mã obj. Nếu FILE không được chỉ rõ thì tên mặc định sẽ là a.out.
-c	Biên dịch nhưng không liên kết.
-DFOO=BAR	Định nghĩa macro tiền xử lý có tên FOO và giá trị BAR trên dòng lệnh.
-IDIRNAME	Báo cho gcc tìm kiếm những file include trong thư mục DIRNAME.
-LDIRNAME	Báo cho gcc tìm kiếm những file thư viện trong thư mục DIRNAME. Mặc định gcc liên kết dựa trên những thư viện dùng chung nằm trong một số thư mục chuẩn như /lib hay /usr/lib.
-static	Liên kết dựa trên những thư viện tĩnh.
-lFOO	Liên kết (link) với thư viện libFOO.
-g	Thêm thông tin gỡ rối (debug) vào mã nhị phân.
-pedantic	In ra tất cả những cảnh báo quy định chuẩn.
-w	Chặn tất cả thông điệp cảnh báo.
-Wall	Thông báo ra tất cả những cảnh báo hữu ích thông thường mà gcc có thể cung cấp.
-werror	Chuyển đổi tất cả những cảnh báo sang lỗi mà sẽ làm ngưng tiến trình biên dịch.
-MM	Đưa ra danh sách phụ thuộc tương thích với make.
-v	Hiện ra tất cả các lệnh đã sử dụng trong mỗi bước của tiến trình biên dịch.

❑ THIẾT LẬP MÔI TRƯỜNG C

- Ưu điểm
  - *Mềm dẻo*
  - *Tính hiệu quả cao*
  - *Có thể chuyển dịch, dễ thích nghi*
  - *Ít từ khóa*
  - *Có cấu trúc module*

- Nhược điểm
  - *Cú pháp khác lạ, khó nhớ*
  - *Một số kí hiệu nhiều nghĩa*
  - *Ngôn ngữ bậc trung, có thể trộn nhiều kiểu dữ liệu*

❑ CÁC VẤN ĐỀ CƠ BẢN

- ***Hàm và thư viện***

- Các chương trình đều dựa trên các hàm để ngăn chúng trở thành những khối mã dài khó đọc hơn, không có những đoạn mã giống hệt nhau lặp đi lập lại.
- Các hàm thư viện là một phần của ngôn ngữ C và có thể xây dựng các hàm và thư viện hàm của riêng mình hoặc của người khác.
- Headers theo chuẩn ISO -C

math.h
- *cos sin tan*
- *exp log pow sqrt*
- *ceil fabs floor fmod*

stdio.h
- *fclose feof fgetpos fopen fread fseek*
- *printf scanf*
- *fgetc fgets fputc fputs getc getchar gets putc putchar puts*

stdlib.h
- *atof atoi atol*
- *calloc free malloc*
- *abort exit getenv system*
- *abs div rand*

string.h
- *strcat strchr strcmp strncmp strcpy strncpy strcspn strlen strstr strtok*

time.h
- *asctime clock difftime time*

❑ CÁC VẤN ĐỀ CƠ BẢN

- ***Hàm và thư viện***

- Heade

Header	FreeBSD 8.0	Linux 3.2.0	Mac OS X 10.6.8	Solaris 10	Description
<aio.h>	•	•	•	•	asynchronous I/O
<cpio.h>	•	•	•	•	cpio archive values
<dirent.h>	•	•	•	•	directory entries
<dlfcn.h>	•	•	•	•	dynamic linking
<fcntl.h>	•	•	•	•	file control
<fnmatch.h>	•	•	•	•	filename-matching types
<glob.h>	•	•	•	•	pathname pattern-matching and generation
<grp.h>	•	•	•	•	group file
<iconv.h>	•	•	•	•	codeset conversion utility
<langinfo.h>	•	•	•	•	language information constants
<monetary.h>	•	•	•	•	monetary types and functions
<netdb.h>	•	•	•	•	network database operations
<nl_types.h>	•	•	•	•	message catalogs
<poll.h>	•	•	•	•	poll function
<pthread.h>	•	•	•	•	threads (
<pwd.h>	•	•	•	•	password file
<regex.h>	•	•	•	•	regular expressions
<sched.h>	•	•	•	•	execution scheduling
<semaphore.h>	•	•	•	•	semaphores
<strings.h>	•	•	•	•	string operations
<tar.h>	•	•	•	•	tar archive values
<termios.h>	•	•	•	•	terminal I/O
<unistd.h>	•	•	•	•	symbolic constants
<wordexp.h>	•	•	•	•	word-expansion definitions

❑ CÁC VẤN ĐỀ CƠ BẢN

- ***Hàm và thư viện***

- Headers theo chuẩn POSIX

<arpa/inet.h>	•	•	•	•	Internet definitions
<net/if.h>	•	•	•	•	socket local interfaces
<netinet/in.h>	•	•	•	•	Internet address family
<netinet/tcp.h>	•	•	•	•	Transmission Control Protocol definitions
<sys/mman.h>	•	•	•	•	memory management declarations
<sys/select.h>	•	•	•	•	select function
<sys/socket.h>	•	•	•	•	sockets interface
<sys/stat.h>	•	•	•	•	file status
<sys/statvfs.h>	•	•	•	•	file system information
<sys/times.h>	•	•	•	•	process times
<sys/types.h>	•	•	•	•	primitive system data types
<sys/un.h>	•	•	•	•	UNIX domain socket definitions
<sys/utsname.h>	•	•	•	•	system name
<sys/wait.h>	•	•	•	•	process control

- ISO C không xử lý các bộ mô tả tệp, đa tiến trình (parent / child) và kiểm soát công việc, định nghĩa không đầy đủ các chức năng đối với hệ thống UNIX …

❑ CÁC VẤN ĐỀ CƠ BẢN

- **Header files**

- Khi gọi một trong các hàm thường yêu cầu phải đưa vào chương trình người dùng một hoặc nhiều file tiêu đề hệ thống khai mà các file đó khai báo hàm sẽ được gọi bao gồm kiểu của nó, các đối số bắt buộc và kiểu của mỗi đối số. Hầu hết các tệp tiêu đề có thể được tìm thấy trong /usr/include hoặc /usr/include/sys.

- Các hướng dẫn thích hợp trong (các) phần cho biết tệp tiêu đề nào cần được đưa vào. Ví dụ, khai báo cho (các) lệnh read(S) hệ thống như sau:

#include <unistd.h>

*ssize_t read (int fildes, void * buf, size t nbyte);*

- Lệnh read(S) muốn dùng thì chương trình phải chứa file tiêu đề *<stduni.h>*.

- Ví dụ khác: trường u_exdata là loại struct được định nghĩa trong *user.h* nằm trong thư mục /usr/include/sys và được đưa vào chương trình bằng cách sử dụng #include *<sys/user. h>*. Đánh dấu điểm xâm nhập (mặc định) của thư viện chương trình

❑ CÁC VẤN ĐỀ CƠ BẢN

- *Hàm **main()***

- Hàm *main()* được hệ điều hành đảm bảo là hàm đầu tiên do người dùng định nghĩa được gọi trong chương trình C trừ khi người lập trình chỉ định rõ ràng một điểm vào chương trình.
- Nguyên mẫu cho hàm **main()** trong hầu hết các môi trường UNIX là:

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

i. Đối số đầu tiên, *argc,* là số các đối số được truyền đến chương trình bao gồm cả tên chương trình.

ii. Các con trỏ đến từng đối số được truyền vào mảng *argv [.]* sao cho *argv[0]* trỏ đến tên của chương trình, *argv[1]* trỏ đến đối số đầu tiên và tiếp tục như vậy cho tới *argv[argc-1]*.

iii. Mảng *envp[.]* chứa các con trỏ đến các biến môi trường shell được truyền cho chương trình. Mỗi biến là một chuỗi có dạng tên = giá trị.

❑ CÁC VẤN ĐỀ CƠ BẢN

Ví dụ: $ cat mains.c

```
#include <stddef.h>
#include <stdio.h>
int main(int argc, char *argv[], char *environ[ ])
{   int x;
    printf("argc = %d \n", argc );
    for (x=0; x<argc; x++) printf("argv[%d] = %s \n", x, argv[x]);
    for (x=0; x<10, x++) printf("environ[%d] = %s \n", x, environ[x]);
   return 0 ;
 }
```

❑ CÁC VẤN ĐỀ CƠ BẢN

```
$ gcc  mains.c –o mains
 $ ./mains mot hai ba bon
    agrc = 5
    argv[0] =   mains
    argv[1] =   mot
    argv[2] =   hai
    argv[3] =   ba
    argv[4] =  bon
```

```
 environ[0] = USERNAME = qtuan
environ[1]  = LANG –c cc
environ[2] = FONTS = /usr/lib/x11/fonts/misci:
                     /usr/lib/x11/fonts/75dpi
environ[3] = VT-100
environ[4] =  PATH = /usr/bin:/bin:/home/qtuan
environ[5] = COLUMMS = 80
environ[6] = LEVEL = wait
environ[7] = WINDOWID = 838469
environ[8] = MAIL = /usr/spool/mail/qtuan
environ[9] = USRLOGIN = false
```

Chú ý: Nội dung biến environ có thể thay đổi phụ thuộc loại shell và nội dung biến

❑ CÁC VẤN ĐỀ CƠ BẢN

- Có hai hàm thư viện để trích xuất và thiết lập các biến môi trường

```
#include <stdlib.h>
    int putenv(char *str);                          Returns: 0 if OK, nonzero on error
    int setenv(const char *name, const char *value, int rewrite);
    int unsetenv(const char *name);                 Both return: 0 if OK, −1 on error
```

- Ví dụ: $ cat unixprompt.c

```
#include <unistd.h>
main(int argc, char *argv[ ])
{   char buf[32] ;
    if (argc != 2)
        exit (-1) ;
     printf("The value of ARG MAX= %d\n", sysconf( SC ARG MAX));
     printf("current value-for PSI = %s\n", getenv("PSI"));
    sprintf(buf, "PS1=%s", argv[l]J ;
    putenv(buf) ;
    printf("new value for PSI = \"%s\•\n", getenv("PSI") );
}
```

Chương trình sử dụng getenv(s) để trích xuất giá trị hiện tại cho biến PS1, in giá trị của nó và sau đó đặt nó thành đối số được truyền đến chương trình bằng cách sử dụng putenv(s). Ban đầu, nó in giá trị của ARG_MAX như được trả về bởi sysconf (S).

❑ CÁC VẤN ĐỀ CƠ BẢN

```
$ cc  unixprompt.c –o unixprompt
 $ ./unixprompt OK>
    The value of ARG MAX= =   102400
    current value-for PSl  = $
    new value for PSl = OK>
 OK>
```

```
$ echo $PS1
$
$ PS1 = "OK >"
$export PS1
OK>
```

- Cần phải thiết lập một trong các biến môi trường của shell trong khi thực hiện một chương trình.

- Lưu ý rằng mỗi khi một tiến trình mới (con) được tạo, một bản sao biến môi trường cha sẽ được tạo ra và bản sao này chỉ thấy khi tiến trình đang chạy. Còn với tiến trình mới, các biến môi trường nằm trong phân đoạn dữ liệu của tiến trình trình nên việc thay đổi nội dung các biến môi trường chỉ thấy đối với tiến trình chúng thay đổi và bất kỳ tiến trình con nào tiếp theo.

❑ CÁC VẤN ĐỀ CƠ BẢN

- *Hàm **exit()***

- Hàm exit(S) và hàm abort(S) - C-ISO là cách tiến trình có thể thoát sau khi rời hàm main(.) tại bất cứ vị trí nào

Khuôn dạng

```
#include <stdlib.h>
void exit(int status);
void _Exit(int status);

#include <unistd.h>
void _exit(int status);
```

Ví dụ:

```
$ cat ex.c
#include <stdlib.h>
int  main(void)
 {   exit(7)   }
$ cc – o ex.c  ex.o
$ ./ex.o
$  echo $?
   7
```

- ISO C định nghĩa _Exit cách để một tiến trình kết thúc mà không cần chạy shell xử lý thoát hoặc shell xử lý báo hiệu

- _exit (POSIX) được gọi bởi exit (C-ISO) và xử lí các chi tiết cụ thể trong HDH Unix

❑ CÁC VẤN ĐỀ CƠ BẢN

- *Hàm **exit()***

- Các hàm có thể có 32 xử lý thoát, nhưng chúng cần được xử lí theo hàng đợi LIFO nhờ lệnh *atexit(S)*

Khuôn dạng:

#include <stdlib.h>

*int atexit(void (*func)(void));*

- Tiến trình *exit(S)* hay *returrn* chỉ được gọi nếu quá trình thoát bình thường ngầm định từ hàm *main()*

- Các tiến trình xử thoát được tạo thông qua việc sử dụng hàm atexit(s). Chương trình với hai trình xử lý thoát, ex1() và ex2(), rồi gọi hàm thư viện exit(s).

```
void  ex1(void)
   { fprintf (stdout,'ex1 được gọi \n '); }
void  ex2(void)
   { fprintf (stdout,' ex2 được gọi \n ');}

main (void)
   {   atexit(ex1) ;
       atexit (ex2) ;
       fprintf (stdout,' Gọi hàm exit \n');
        exit(0)
   }

$ ./ex.o
Gọi hàm exit
ex2 được gọi
ex1 được gọi
```

❑ CÁC VẤN ĐỀ CƠ BẢN

- Biến, kiểu dữ liệu, hàm số học

Type	Constant Examples	printf chars
char	'a', '\n'	%c
_Bool	0, 1	%i, %u
short int	—	%hi, %hx, %ho
unsigned short int	—	%hu, %hx, %ho
int	12, -97, 0xFFE0, 0177	%i, %x, %o
unsigned int	12u, 100U, 0XFFu	%u, %x, %o
long int	12L, -2001, 0xffffL	%li, %lx, %lo
unsigned long int	12UL, 100ul, 0xffeeUL	%lu, %lx, %lo
long long int	0xe5e5e5e5LL, 50011	%lli, %llx, &llo
unsigned long long int	12ull, 0xffeeULL	%llu, %llx, %llo
float	12.34f, 3.1e-5f, 0x1.5p10, 0x1P-1	%f, %e, %g, %a
double	12.34, 3.1e-5, 0x.1p3	%f, %e, %g, %a
long double	12.341, 3.1e-5l	%Lf, $Le, %Lg

❑ CÁC VẤN ĐỀ CƠ BẢN

- Biến, kiểu dữ liệu, hàm số học

- Về cơ bản giống như các ngôn ngữ C trên các môi trường

$ cat bsh.c

```
#include <stdio.h>
int main (void)
{
  int integerVar = 100;
  float floatingVar = 331.79;
 double doubleVar = 8.44e+11;
 char charVar = 'W';
 _Bool boolVar = 0;
  printf ("integerVar = %i\n", integerVar);
  printf ("floatingVar = %f\n", floatingVar);
  printf ("doubleVar = %e\n", doubleVar);
  printf ("doubleVar = %g\n", doubleVar);
  printf ("charVar = %c\n", charVar);
 printf ("boolVar = %i\n", boolVar); return 0;
}
```

❑ CÁC VẤN ĐỀ CƠ BẢN

- Biến, kiểu string

- Một thuộc tính thiết yếu chuỗi ký tự là chuỗi các phần tử ký tự riêng lẻ có độ dài không xác định. Hầu hết các ký tự riêng lẻ có thể nhập vào bàn phím được biểu diễn bằng mã ASCII: kiểu dữ liệu C **char** được sử dụng để lưu trữ dữ liệu ký tự.

- Các chuỗi lưu trữ i dạng mảng các ký tự kết thúc bằng NULL vì vậy mảng phải đủ lớn để chứa chuỗi ký tự cộng thêm một. Hãy nhớ rằng mảng luôn được count từ 0.

```
#include <stdio.h>
int main(int argc, char *argv[], char *env[])
{    char c1 = 'd';
     char c2 = 'a';
     char c3 = 'v';
     char c4 = 'i';
     char name[5] = "";
     sprintf(name,"%c%c%c%c",c1,c2,c3,c4);
     printf("%s\n",name);
    return 0;
}
```

- Các kiểu biến UNIX – POSIX khác với ISO C

clock_t	*Đếm ticks clock (thời gian tiến trình)*
comp_t	*ticks clock thu gọn (not defined by POSIX.1;*
dev_t	*Số thiết bị (lớn và nhỏ)*
fd_set	*Các tập mô tả file*
fpos_t	*Vị trí file*
gid_t	*Số group IDs*
ino_t	*Số i-node*
mode_t	*Loại file, mode tạo file*
nlink_t	*Đếm liên kết với thư mucj*
off_t	*Kích thước và độ trượt file(theo dấu)*
pid_t	*IDs tiến trình và IDs tiến trình (signed)*
pthread_t	*thread IDs*
ptrdiff_t	*Kết quả của phép trừ hai con trỏ (signed)*
rlim_t	*Các gới hạn tài nguyên*
sig_atomic_t	*Loại dữ liệu có thể được truy cập sâu*
sigset_t	*Tập báo hiệu*
size_t	*Kích thước của objects (such as strings)*
ssize_t	*Hàm trả lại số byte được đếm (signed)*
time_t	*Thời gian calendar tính đến giây*
uid_t	*Số IDs gười dùng*
wchar_t	*Có thể biểu diễn tất cả các đặc tín riêng biệt*

❑ CÁC VẤN ĐỀ CƠ BẢN

- Các kiểu biến UNIX – POSIX khác với ISO C

```
#include <stdio.h>
int scanf(const char *restrict format, ...);
```

Xử lí: Phân tích cú pháp một chuỗi đầu vào và chuyển đổi chuỗi ký tự thành các biến có kiểu cụ thể. Các đối số theo sau định dạng chứa địa chỉ của các biến để khởi tạo với kết quả của các chuyển đổi.

Dấu (%) chỉ định dang của một đặc tả cần chuyển đổi. Ngoại trừ thông số kỹ thuật chuyển đổi và khoảng trắng, các ký tự định dạng phải khớp với đầu vào. Nếu một ký tự không khớp, quá trình xử lý sẽ dừng lại, phần còn lại của dữ liệu nhập chưa được đọc.

```
#include <unistd.h>
ssize_t read(int fd, void *buf, size_t nbytes);
```

Returns: number of bytes read if OK, 0 if end of file, −1 on error

Xử lí - Kích thước đọc từ tệp thông thường, nếu đến cuối tệp trước khi số byte yêu cầu được đọc.

- Đọc từ thiết bị đầu cuối. Thông thường, tối đa một dòng được đọc cùng một lúc.

- Đọc từ mạng. Việc lưu vào bộ đệm trong mạng có thể ít hơn số yêu cầu.

- Đọc từ đường ống / FIFO. Nếu đường ống chứa ít byte hơn, read sẽ trả về những gì có.

- Đọc từ một thiết bị - chẳng hạn như băng từ, có thể trả về một bản ghi cùng một lúc.

- Khi bị gián đoạn bởi một báo hiệu và một phần dữ liệu đã được đọc.

❑ CÁC VẤN ĐỀ CƠ BẢN

- **Biến cấu trúc**

- Mảng cho phép nhóm các phần tử cùng kiểu thành một thực thể logic. Để tham chiếu đến một phần tử trong mảng, tất cả những gì cần thiết là tên của mảng được đưa ra cùng với chỉ số con thích hợp.
- Ví dụ: Có thể dựa trên ba biến để xác định date

```
int month = 9, day = 25, year = 2004;
```

Tuy nhiên biến date với khai báo

```
struct date
{ int month;
  int day;
  int year; };
```

Khi sử dụng

```
date.day = 25 ;
date.month = 9 ;
date,year = 2004 ;
```

❑ CÁC VẤN ĐỀ CƠ BẢN

- **Các trường exdata**

Một con trỏ inode được sử dụng để truy cập thiết bị.

Các trường exdata		*Mô tả*
struct inode	**ip*	*Inode with which the executable data is associated*
long	*ux tsize*	*Kích thước code tiến trình.*
long	*ux dsize*	*Kích thước data.*
long	*ux bsize*	*The BSS size.*
long	*ux lsize*	*The library size*
long	*ux nshlibs*	*Số libraries cần chia xẻ*
long	*ux toffset*	*File offset tới code*
long	*ux_doffset*	*File offset tới data.*
long	*ux loffset*	*File offset tới phần chứa các tên library*
long	*ux_txtorg*	*Địa chỉ bắt đầu của code trong bộ nhớ*
long	*ux_datorg*	*Địa chỉ bắt đầu của data trong bộ nhớ.*
long	*ux entloc*	*Địa chỉ mà chương trình sẽ có.*

❑ CÁC VẤN ĐỀ CƠ BẢN

- **Biến con trỏ**

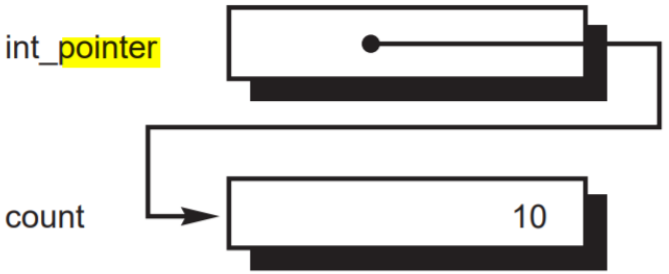
- Con trỏ cho phép biểu diễn hiệu quả các cấu trúc dữ liệu phức tạp, thay đổi các giá trị được truyền dưới dạng đối số cho các hàm, để làm việc với bộ nhớ đã được cấp phát "động“
- Con trỏ là phương tiện *gián tiếp* để truy cập giá trị của mục dữ liệu cụ thể.

Ví dụ: *Biến count có thể được sử dụng*

Biến count được truy cập gián tiếp

```
int count = 10;
int *int_pointer
int_pointer = &count
```

```
#include <stdio.h>
int main (void)
{    int count = 10, x;
     int *int_pointer;
     int_pointer = &count;
     x = *int_pointer;
     printf ("count = %i, x = %i\n", count, x);
     return 0;
}
```



❑ CÁC VẤN ĐỀ CƠ BẢN

- **Biến con trỏ**

- *Pointer* cũng có thể được định nghĩa để trỏ đến *struct*.

- Xét biến

```
struct date
{   int month;
    int day;
    int year; };
```

- Khi khởi tạo có thể khởi tạo như vậy gán giá trị hoặc

```
struc date  todaysDate ;
struct date *datePtr ;
datePtr = &todaysDate ;
(*datePtr).day = 25;
datePtr -> day = 25
```

```
#include <stdio.h>
int main (void)
{   struct date
    { int month;
      int day;
      int year; };
   struct date today, *datePtr;
    datePtr = &today;
    datePtr->month = 9;
    datePtr->day = 25;
    datePtr->year = 2004;
 printf ("Today's date is %i/%i/%.2i.\n", datePtr-
>month, datePtr->day, datePtr->year % 100);
return 0;
}
```

- Lệnh vào /ra – ISO C

❑ CÁC VẤN ĐỀ CƠ BẢN

- Họ *scanf()* phân tích cú pháp chuỗi đầu vào, và chuyển đổi chuỗi ký tự thành các biến có kiểu cụ thể, hàm fscanf () được sử dụng để đọc tập hợp các ký tự từ file, còn sscanf đọc input đã được định dạng từ một chuỗi string

```
#include <stdio.h>
int scanf(const char *restrict format, ...);
int fscanf(FILE *restrict fp, const char *restrict format, ...);
int sscanf(const char *restrict buf, const char *restrict format, ...);
```

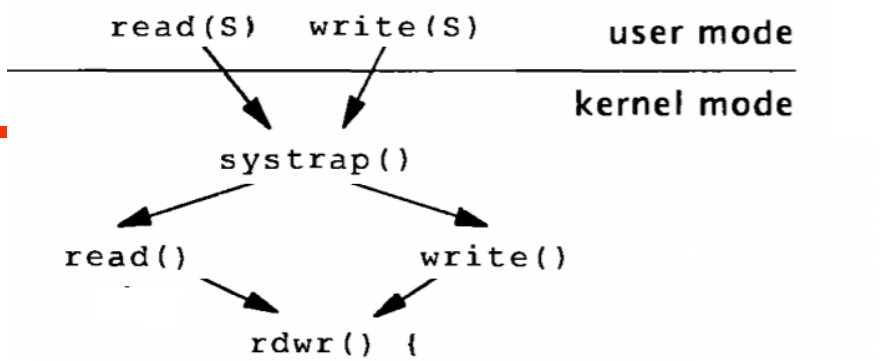
EOF if input error or end of file before any conversion

- Hàm *printf()* ghi vào đầu ra tiêu chuẩn, fprintf ghi vào luồng được chỉ định, dprintf ghi vào bộ mô tả tệp được chỉ định và sprintf đặt các ký tự được định dạng trong mảng buf. Hàm sprintf tự động thêm byte null vào cuối mảng, nhưng byte null này không được bao gồm trong giá trị trả về.

```
#include <stdio.h>
int printf(const char *restrict format, ...);
int fprintf(FILE *restrict fp, const char *restrict format, ...);
int dprintf(int fd, const char *restrict format, ...);
```

Negative any error

C Programming



❑ CÁC VẤN ĐỀ CƠ BẢN

- Lênh read(S)/write(S)

- Khuôn dạng

```
#include  <unistd.h>
ssize_t  read(int fildes, void *buf, size t nbyte)
ssize_t write(int fildes, const void *buf, size t nbyte)
```

Và

```
#include <sys/types.h>
#include <sys/uio.h>
ssize_t readv(int fildes, const struct iovec *iov, int iovcnt) ;
ssize_t writev(int fildes, const struct iovec *iov, int iovcnt) ;
```

Trong đó: + readv(S) đọc nội dung file vào trong nhiều bộ nhớ đệm (buffer)

+ writev(S) ghi file khi dùng số bộ đệm được chỉ định.

- Cơ chế I/0 không phải là một phần của ngôn ngữ C, nhưng chuẩn đầu vào, đầu ra và lỗi được sử dụng trong C/UNIX và biểu thị bằng ký hiệu *stdin, stdout và stderr*.

File descriptor	File stream	Description
0	stdin	Standard input.
1	stdout	Standard output.
2	stderr	Standard error.

❑ CÁC VẤN ĐỀ CƠ BẢN

- Ví dụ

```
#include <signal.h>
extern char *signm[]

void  sighdlr(int signo, siginfo_t *si, void *uctxt)
{     printf ("caught %s\n", signm[signo - 1)) ; }

 main(int argc, char *argv[])
{     struct sigaction act ;
      char buf[256) ;
      int n ;
      int restart ( argc == 2 & & argv [1] == 'r' );
       ....
      printf("PID = %d \n", getpid() );
      n = read(0, buf, 256);
      if (n == -1)
         pexit("read") ;
      else {
        buf[n] = '\0' ;
        printf("DATA = %s ", buf); }
 }
```

❑ CÁC VẤN ĐỀ CƠ BẢN

- Hàm vòng lặp/điều kiện *Sinh viên tự đọc các lệnh vòng lặp, điều khiển luồng*

✓ for(initial value; keep on until ; incremental change)

{ do this; and this; and this; }

✓ while(this is true)

{ do this; and this; and this; }

✓ switch(on an integer or character value)

```
{     case 0: do this; and this; and this; break;
      case n: do this; and this; and this; break;
      default: do this; and this; and this; break;     }
```

✓ if(this is true)

{ do this; and this; and this; }

else if (this is true)

{ do this; and this; and this; }

else { do this; and this; and this; }

❑ CÁC VẤN ĐỀ CƠ BẢN

- Ví dụ

```
#include <stdio.h>
#include <string.h>
#include <math.h>
double doit(int number1, int number2)
{
    return sqrt((double)(number1 + number2) );
}
int main(int argc, char *argv[], char *env[])
{   int n1 = 0, n2 = 0, i=0;
    n1 = atoi((char *) strtok(argv[1],":"));
    n2 = atoi((char *) strtok(NULL,":"));
    printf("Content-type:\n");
    for(i=1;i<=100;i++)
       printf("%f ",doit(n1+i,n2*i));
    printf("\n");
    return 0;
}
```

- Hàm atoi chuyển đổi biểu diễn chuỗi của một số nguyên thành một số nguyên ("1" thành 1).
- Hàm strtok gọi với tên chuỗi mà bạn muốn phân tích cú pháp, sau đó trong các lần gọi tiếp theo, tham số đầu tiên được thay đổi thành NULL.

❑ CÁC VẤN ĐỀ CƠ BẢN

- **HÀM BÁO LỖI**

- Khi có một lỗi xảy ra các hàm trong hệ thống Unix thì hàm lỗi sẽ trả giá trị -1 báo có lỗi và số nguyên thông báo loại lỗi tìm thấy trong <sys/errno.h>

+ Chương trình bên cố gắng mở một tệp không tồn tại để hiển thị giá trị lỗi

+ Lệnh *perror(const char *s)* có thể được sử dụng để hiển thị thông báo mô tả lỗi được liên kết với giá trị trong errno. Chuỗi được chuyển cho (các) perror luôn được hiển thị đầu tiên.

```
#include <fcntl.h>
#include <stdio.h>
extern int errno ;
main()
{
    int fd ;
    fd = open("This file_does_not_exist", O_RDONLY) ;
   if (fd == -1)
   {
        printf ("fd = %d, errno = %d\n", fd, errno) ;
        perror("errno meaning") ;
    }
}
```

❑ CÁC VẤN ĐỀ CƠ BẢN

- **HÀM BÁO LỖI**

- Kí hiệu lỗi bắt đầu chữ E và các hằng số (*Xem Unix manuals*)

Ví dụ: $ cat loi.c

```
# include "apue.h"
#include <errno.h>
int main(int argc, char *argv[])
{     printf("EACCES: %s\n", strerror(EACCES));
       errno = ENOENT;
        perror(argv[0]);
        exit(0);
 }
```

- Sau khi dich ra loi.o và chay

```
$ ./loi.o
 EACCES: Permission denied .
 /a.out: No such file or directory
```

E2BIG: Danh sách argument quá dài
EACESS: Cố truy cập file bị cấm
EALREADY: Hoạt động tại đối tượng đã làm
EBADF: filedes không phù hợp khi mở file
EBADFD: … hoặc đọc file chỉ viết
ECHILD: wait(S) vận hành khi không có child
EDEADLK: deadlock được detect và từ chối
EEXITS: file hiện hành bị bỏ qua
EFAULT: phát hiện phần cứng lỗi
EFBIG: file vượt quá kích thước
EINTR: Lỗi ngắt
EINVAL: argument không được chấp nhận
EIO: Lỗi vào ra
EISRIR: không mở được
ELOOP: Nhiều symbollink trong pathname
EMFILE: Số mô tả file mở quá OPEN_MAX
ENFILE: *Bảng file hệ thống bị đầy*
ENODEV: *Làm trên thiết bị không thích hợp*
ENOENT: *Tên file chỉ định không tồn tại ..*
……

❑ CÁC VẤN ĐỀ CƠ BẢN

- *Hàm **errno()***

- Ví dụ:

```
#include <stdio.h>
#include <errno.h>
main (int argc, char *argv[])
{     FILE *fp;
      fp = fopen(argv[1], "r");
      if (fp==NULL)
          {   perror(argv[0]);
          perror(NULL);
          perror("File could not be opened");
          printf("errno = %d\n", errno);
          exit(1);      }
      printf("File exists and is opened for reading ");
      fclose(fp);   exit(0);
}
```

```
$ ./demoperror
      ./demoperror: Bad address
      Bad address
      File could not be opened: Bad address
      errno = 14
$ ./demoperror bank.lst
      ./demoperror: No such file or directory
      No such file or directory
      File could not be opened: No such file or directory
      errno = 2
$ ./demoperror xyz.txt
  File exists and is opened for reading
```

❑ CÁC VẤN ĐỀ CƠ BẢN

- **TIME VALUEs**

- UNIX duy trì *hai giá trị thời gian* khác nhau:

1. Thời gian lịch: sử dụng để ghi lại thời gian tệp được sửa đổi lần cuối. Kiểu dữ liệu *time_t* chứa các giá trị thời gian này.

2. Thời gian xử lý: là thời gian CPU được sử dụng bởi một tiến trình đo trước đây là 50, 60 hoặc 100 tích tắc mỗi giây. Kiểu dữ liệu hệ thống nguyên thủy *clock_t* . Hệ thống UNIX duy trì ba giá trị cho một tiến trình:

• Đồng hồ thời gian • Thời gian CPU của người dung • Thời gian CPU hệ thống

- Trong shell, lệnh time(1) cho ta *biết* time *clock, user time* và *system time*

```
$ cd /usr/include
$ time -p grep _POSIX_SOURCE */*.h  >  /dev/null
      real  0.81s
      user 0.11s
      sys   0.07s
```

```
#include <sys/times.h>
clock_t times(struct tms *buf );

Returns: elapsed wall clock time in clock ticks if OK, −1 on error
```

C Programming

- **TIME VALUEs**

```
struct tms {
        clock_t tms_utime;
        clock_t tms_stime;
        clock_t tms_cutime;
        clock_t tms_cstime;
};
```

```
static void do_cmd(char *cmd) /* execute and time the "cmd" */
{
        struct tms tmsstart, tmsend;
        clock_t start, end;
        int status;
        printf("\ncommand: %s\n", cmd);
        if ((start = times(&tmsstart)) == -1) /* starting values */
                err_sys("times error");
         if ((status = system(cmd)) < 0) /* execute command */
                err_sys("system() error");
        If ((end = times(&tmsend)) == -1) /* ending values */
                 err_sys("times error");
          pr_times(end-start, &tmsstart, &tmsend);
          pr_exit(status);
}
```

```
#include "apue.h"
#include <sys/times.h>
static void pr_times(clock_t, struct tms *, struct tms *);
static void do_cmd(char *);
Int  main(int argc, char *argv[])
{    int i;
     setbuf(stdout, NULL);
      for (i = 1; i < argc; i++)
           do_cmd(argv[i]); /* each command-line arg */
      exit(0);
}
```

❑ **CÁC VẤN ĐỀ CƠ BẢN**

❑ CÁC VẤN ĐỀ CƠ BẢN

- **TIME VALUEs**

```
static void pr_times(clock_t real, struct tms *tmsstart, struct tms *tmsend)
{
    static long clktck = 0;
     if (clktck == 0)                /* fetch clock ticks per second first time */
         if ((clktck = sysconf(_SC_CLK_TCK)) < 0)
              err_sys("sysconf error");
          printf(" real: %7.2f\n", real / (double) clktck);
    printf(" user: %7.2f\n",(tmsend->tms_utime - tmsstart->tms_utime) / (double) clktck);
    printf("sys: %7.2f\n", (tmsend->tms_stime - tmsstart->tms_stime) / (double) clktck);
    printf("child user: %7.2f\n",(tmsend->tms_cutime - tmsstart->tms_cutime)/(double) clktck);
    printf(" child sys: %7.2f\n",(tmsend->tms_cstime - tmsstart->tms_cstime)/(double) clktck);
}
```

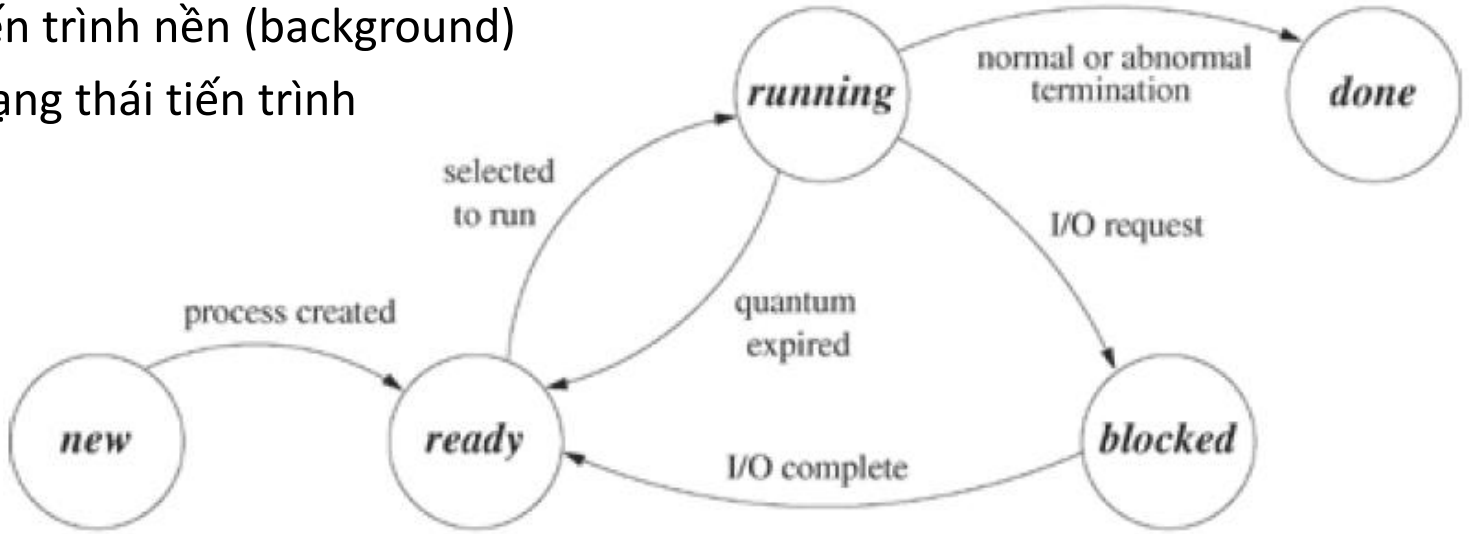
```
$ ./a.out "sleep 5" "date"
command: sleep 5
real: 5.01
user: 0.00
sys: 0.00
child user: 0.00
child sys: 0.00
normal termination, exit status = 0
```

```
command: date
Sun Feb 26 18:39:23 EST 2012
real: 0.00
user: 0.00
sys: 0.00
child user: 0.00
child sys: 0.00
normal termination, exit status = 0
```

❑ CÁC VẤN ĐỀ CƠ BẢN

- **TIẾN TRÌNH**

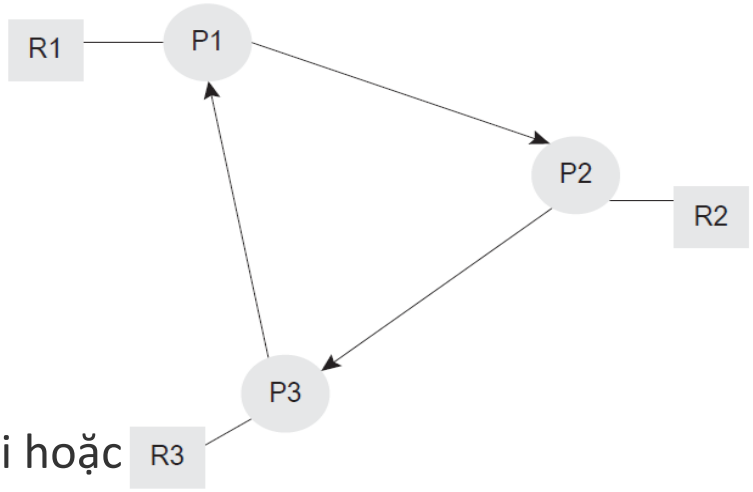
- Hai loại tiến trình người dùng xây dựng trên C i) tiến trình mặt (forceground) và tiến trình nền (background)
- Trạng thái tiến trình



State	*meaning*
new	*Đang tạo*
running	*Lệnh đang vận hành*
blocked	*Đang đợi một sự kiện (ví dụ I/O)*
ready	*Đang đợi để được gán vào một tiến trình*
done	*Xong*

❑ CÁC VẤN ĐỀ CƠ BẢN

- **TIẾN TRÌNH**



- **Deadlock** là tình huống ứng dụng bị treo do hai hoặc nhiều tiến trình không thể tiếp tục vì mỗi tiến trình đang chờ một trong những tiến trình khác giải phóng khóa trên một số vùng nhất định.

- Giả sử rằng có ba tiến trình đang chạy P1, P2 và P3, đã nhận được khóa ghi lần lượt trên các vùng R1, R2 và R3. Theo trình tự xử lí, tiến trình P1 muốn quá trình P2 giải phóng khóa ghi trên vùng R2. và tương tự tiến trình P2 muốn quá trình P3 giải phóng khóa ghi trên vùng R3. Tuy nhiên tiến trình P3 muốn quá trình P1 giải phóng khóa ghi trên vùng R1 … do đó, có một vòng tiến trình trong đó mỗi tiến trình đang chờ một quá trình khác giải phóng khóa trên một vùng để tiến hành. Do đó, không có tiến trình nào có thể hoàn thành nhiệm vụ của nó và bị treo trong một thời gian vô hạn được gọi là bế tắc.

❑ CÁC VẤN ĐỀ CƠ BẢN

- **TIẾN TRÌNH**

- Nguyên nhân deadlock: i) **Loại trừ lẫn nhau** ii) **Giữ và chờ** iii) **Không có quyền ưu tiên** và iv) **Chờ theo vòng tròn**.

- Cách để giải quyết deadlock:

a) Phát hiện và phục: Tiến hành trước một tiến trình và giải phóng các khóa mà nó có được. deadlock được phát hiện bằng cách lập lịch phân bổ/yêu cầu tài nguyên và kiểm tra các chu kỳ. Tránh tạo chu kỳ bằng cách thực hiện thuật toán chu trình.

b) Ngăn chặn: Kiểm tra trước khi gán các khóa vùng. Nếu việc cấp khóa trên một vùng có thể dẫn đến bế tắc, thì không nên cấp.

c) Bỏ qua: giả định deadlock sẽ không bao giờ xảy ra.

d) Tránh: hệ điều hành biết trước yêu cầu tài nguyên của tất cả các tiến trình trong suốt thời gian tồn tại của chúng, hệ điều hành sẽ kiểm tra lần đầu và đảm bảo rằng việc phân bổ tài nguyên sẽ không dẫn đến bế tắc –sử dụng **Thuật toán Banker,**

Thanks