C program

dqmdang@hcmus.edu.vn

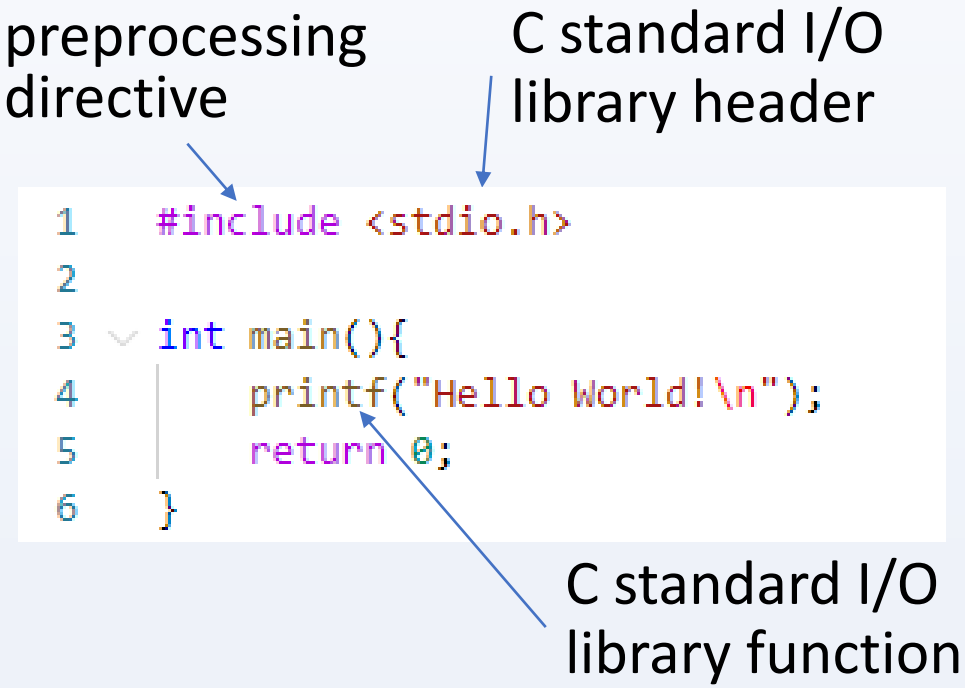
Nội dung

Bài học trước giới thiệu về hệ điều hành Linux và các command cơ bản để làm việc với Linux. Đa số các command đó là program viết bằng ngôn ngữ C, đối tượng tìm hiểu của bài học hôm nay:

- C program
- C library function và system call
- Build and run C program
- Làm việc với file trên C
- Làm việc với process trên C

C program

“Hello World”



- Chương trình C bắt đầu thực thi từ hàm main()
- Theo quy ước, hàm main() return 0 nếu không có lỗi xảy ra và return !=0 nếu có lỗi (thường là mã lỗi có giá trị từ 1 => 255)

“echo ?”

```c
#include <stdio.h>

int main(int argc, char *argv[])
{
    for (int i = 0; i < argc; i++)
    {
        printf("%s ", argv[i]);
    }
    printf("\n");
    return 0;
}
```

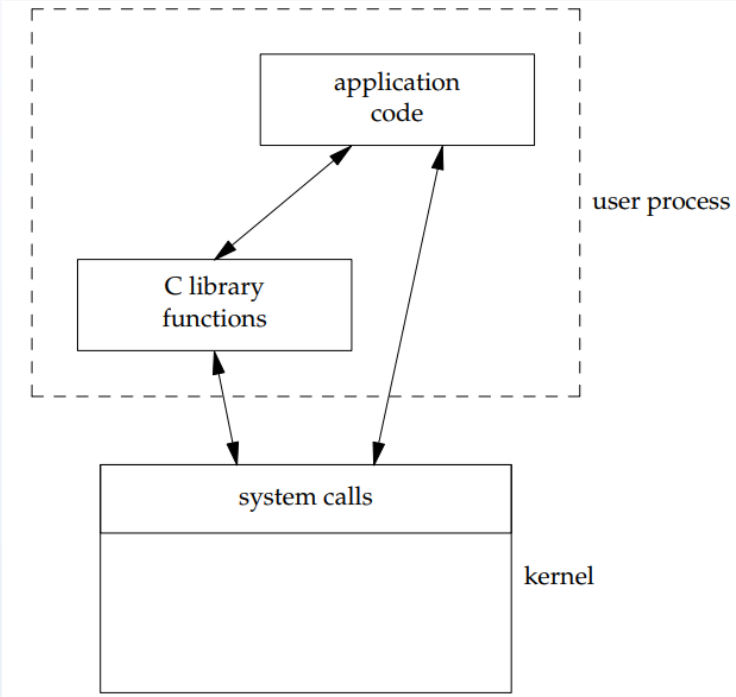
- Hàm main() có thể nhận tham số khi khai báo với prototype:

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

- Trong đó argc là số lượng tham số, argv là mảng của các con trỏ trỏ tới các tham số theo định dạng chuỗi kí tự
- Theo quy ước argv[0] là tham số chứa tên command

C library và System call

- System call là giao diện lập trình cung cấp các service của kernel. System call tuân theo quy chuẩn ABI đối với mỗi platform. Do đó cần sử dụng assembly để gọi trực tiếp System call
- C library cung cấp lớp wrapper cho các System call và hỗ trợ thêm các function ở mức chức năng cao hơn



“Hello World” sử dụng System call wrapper

```c
#include <unistd.h>

int main()
{
    const char string[] = "Hello, World!\n";
    write(1,string,sizeof(string)-1);
    _exit(0);
}
```

- Chương trình “Hello world” gọi System call thông qua wrapper của C library
- Hầu hết các System call đều có wrapper định nghĩa trong header <unistd.h>
- Build chương trình này với câu lệnh:

gcc hw.c -o hw.out

“Hello World” sử dụng syscall function

```c
#define _DEFAULT_SOURCE
#include <unistd.h>
#include <sys/syscall.h>   /* For SYS_xxx definitions */

int main()
{
    const char string[] = "Hello, World!\n";
    syscall(SYS_write,1,string,sizeof(string)-1);
    syscall(SYS_exit,0);
}
```

- Chương trình “Hello world” gọi System call thông qua *syscall* function
- Sử dụng syscall function cần kiến thức về ABI của platform. Do đó nó chỉ được dùng nếu một System call không có wrapper
- Build chương trình này với câu lệnh:

gcc hw.c -o hw.out

“Hello World” i386 assembly

```
.section .text
.globl _start
_start:
    movl    $len,%edx           # third argument: message length
    movl    $msg,%ecx           # second argument: pointer to message
    movl    $1,%ebx             # first argument: file descriptor (stdout)
    movl    $4,%eax             # system call number (sys_write)
    int     $0x80               # call kernel

    movl    $0,%ebx             # first argument: exit code
    movl    $1,%eax             # system call number (sys_exit)
    int     $0x80               # call kernel

.section .data
msg:
    .ascii "Hello, World!\n"    # our dear string
    len = . - msg               # length of our dear string
```

- Chương trình “Hello world” viết bằng assembly gọi trực tiếp System call trên i386
- Build chương trình này với câu lệnh:

  gcc -static hw.s -nostdlib -o hw.out

“Hello World” x86_64 assembly

```
.section .text
.globl _start
_start:
    movq    $len,%rdx           # third argument: message length
    movq    $msg,%rsi           # second argument: pointer to message
    movq    $1,%rdi             # first argument: file descriptor (stdout)
    movq    $1,%rax             # system call number (sys_write)
    syscall                     # call kernel

    movq    $0,%rdi             # first argument: exit code
    movq    $60,%rax            # system call number (sys_exit)
    syscall                     # call kernel

.section .data
msg:
    .ascii "Hello, World!\n"    # our dear string
    len = . - msg               # length of our dear string
```

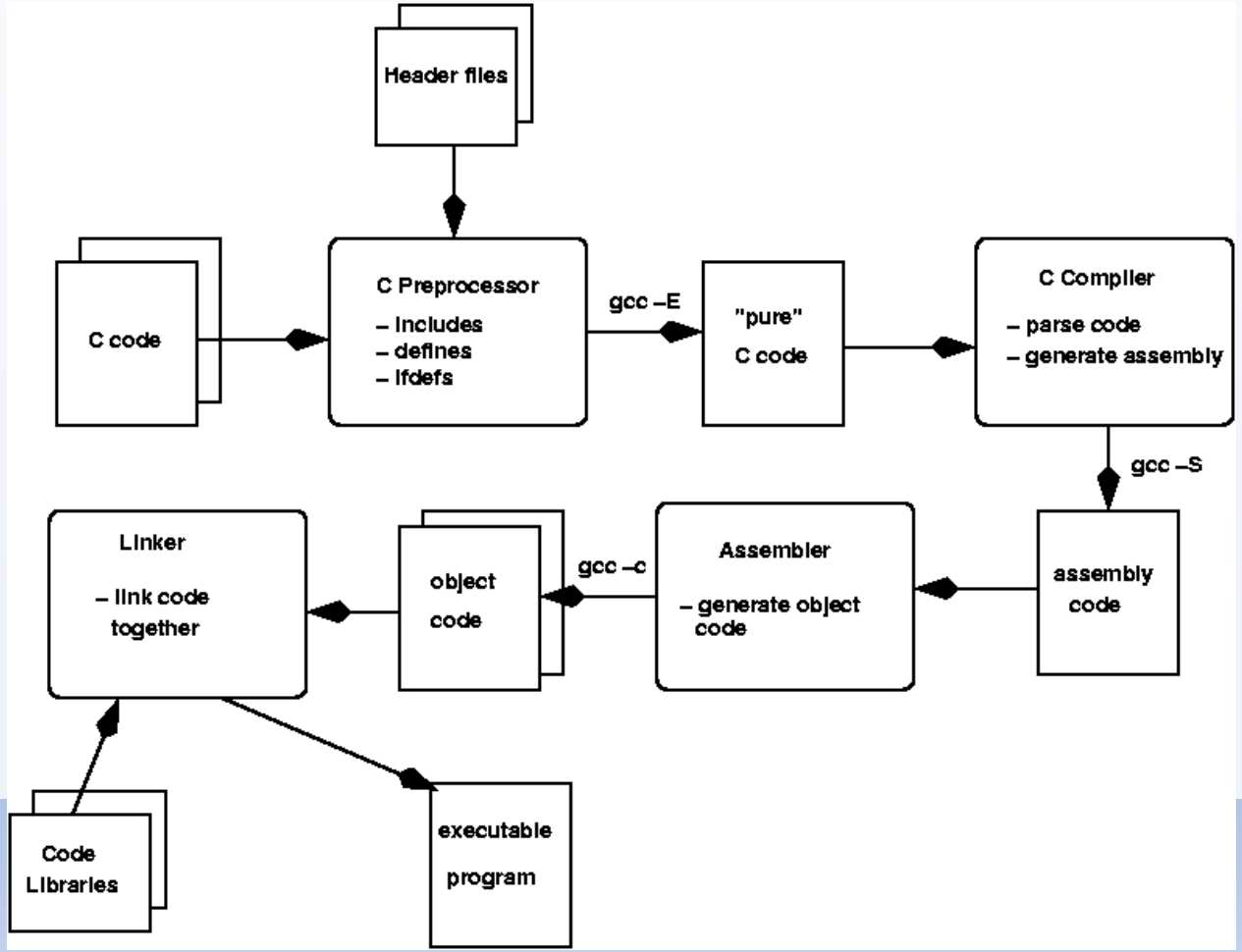
- Chương trình “Hello world” viết bằng assembly gọi trực tiếp System call trên x86_64
- Build chương trình này với câu lệnh:

  gcc -static hw.s -nostdlib -o hw.out

C library và system call

- glibc là C library phổ biến trên các hệ điều hành Linux
- Thông tin về các system call và các function trong glibc được tra cứu từ section 2 và 3 của command *man*
- Các system call sử dụng bởi một process có thể được truy vết bằng command *strace*
- Các function trong C library thường return -1 hoặc NULL khi xảy ra lỗi. Nguyên nhân gây ra lỗi được lưu trữ trong biến (hoặc macro) *errno*

Build program from source code



Quá trình tạo program từ source code C:

- Preprocess
- Compile
- Assembly
- Link

gcc

- gcc (GNU Compiler Collection) là compiler phổ biến trên các hệ điều hành Linux
- Compile các file source code C bằng command:

```
gcc <file1.c> <file2.c> <...> -I <header dir> -o <output file name>
```

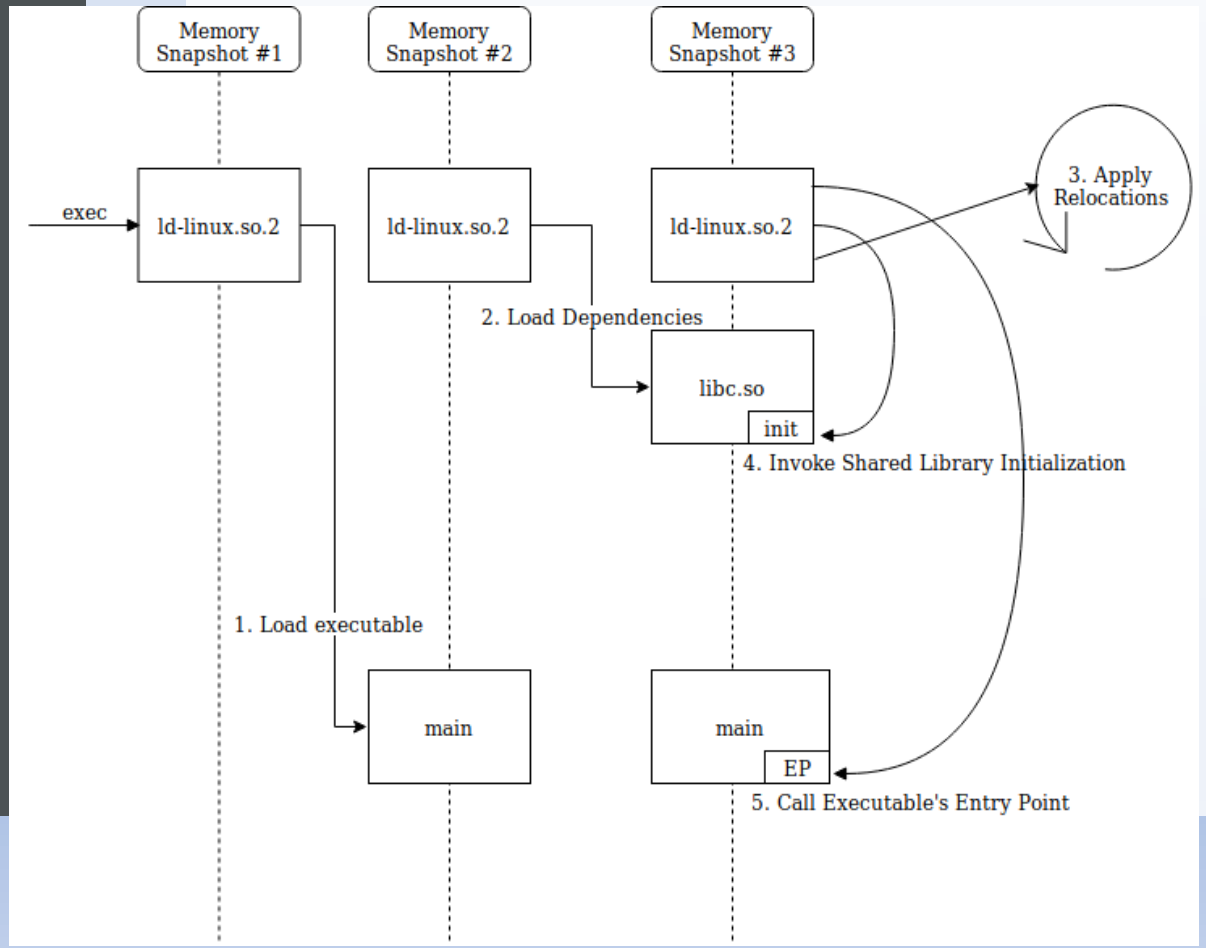
- Theo mặc định gcc link program với shared library (dynamic link), để link với static library (static link), cần thêm option -static vào command trên
- Kết quả của quá trình build là object file

Object file

Linux sử dụng định dạng elf cho các object file. Sử dụng các command sau để xem thông tin và nội dung của một object file

- file <obj>: xem định dạng file
- xxd -g 1 <obj>: xem file dưới dạng các byte
- readelf -a <obj>: xem thông tin header của file
- objdump -dr <obj>: xem disassembly của file
- nm <obj> : xem các symbols có trong file
- size <obj>: liệt kê kích thước của các segment trong file
- ldd <obj>: xem các thư viện phụ thuộc (shared library)

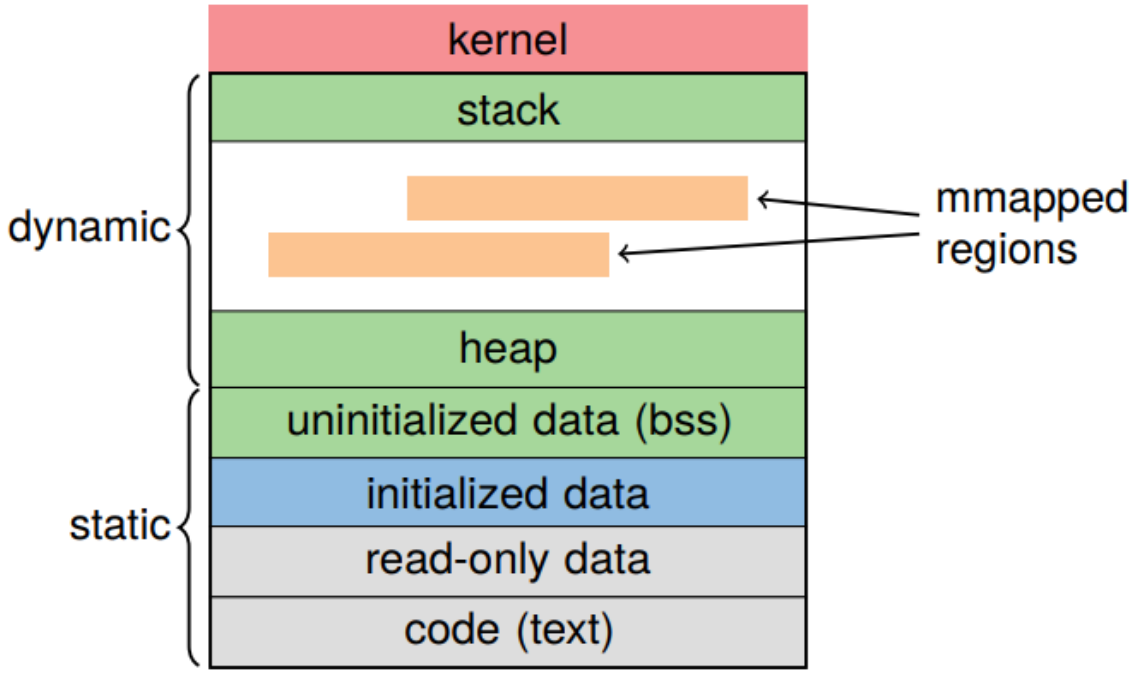
Run program



Quá trình run program:

- Load program
- Load library
- Dynamic relocation
- Init library
- Run program at entry point

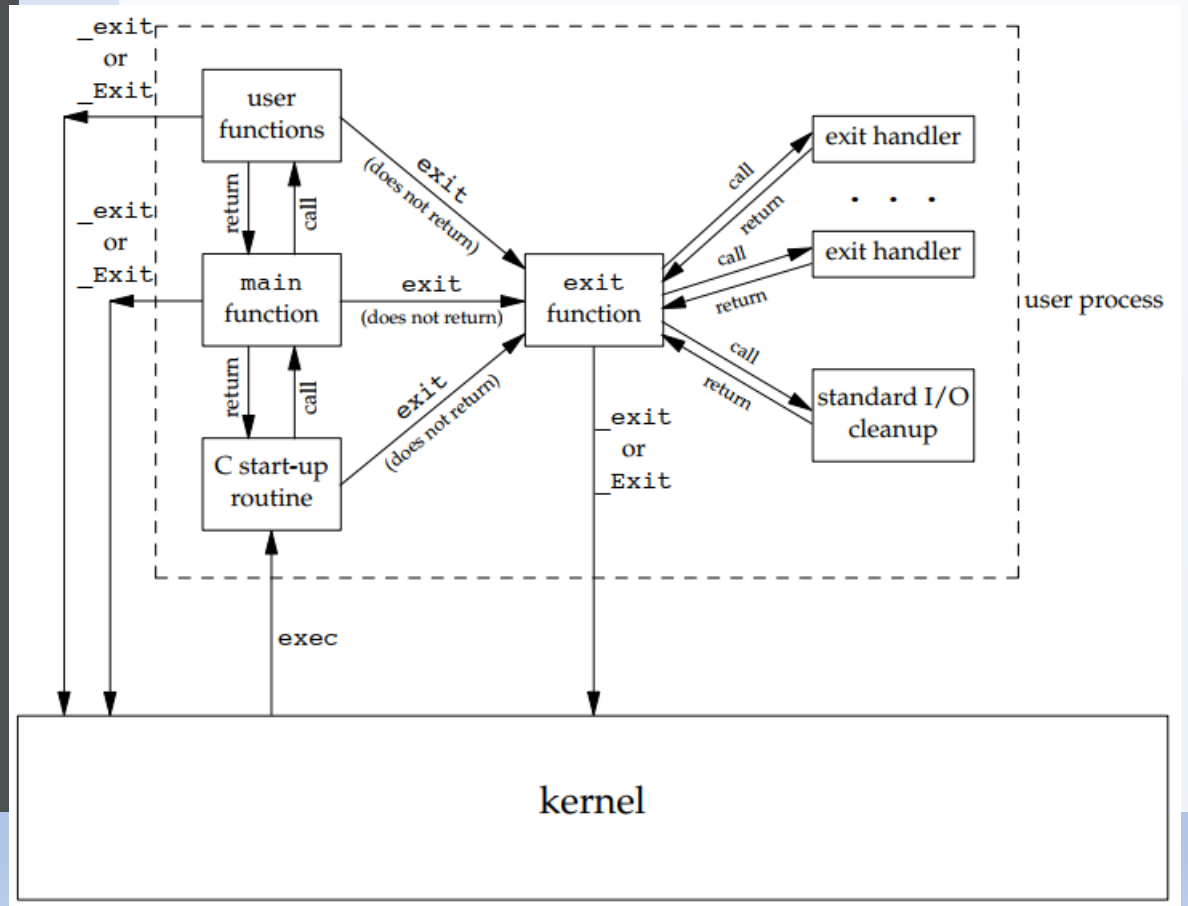
Processs virtual address space



Address space divided into “segments”:

- Heap: allocated and laid out at runtime by malloc
- Stack: allocated at runtime (procedure calls), layout by compiler
- Global data/code: allocated by compiler, layout by linker
- Mmapped regions: Managed by programmer or linker

C program start and terminate



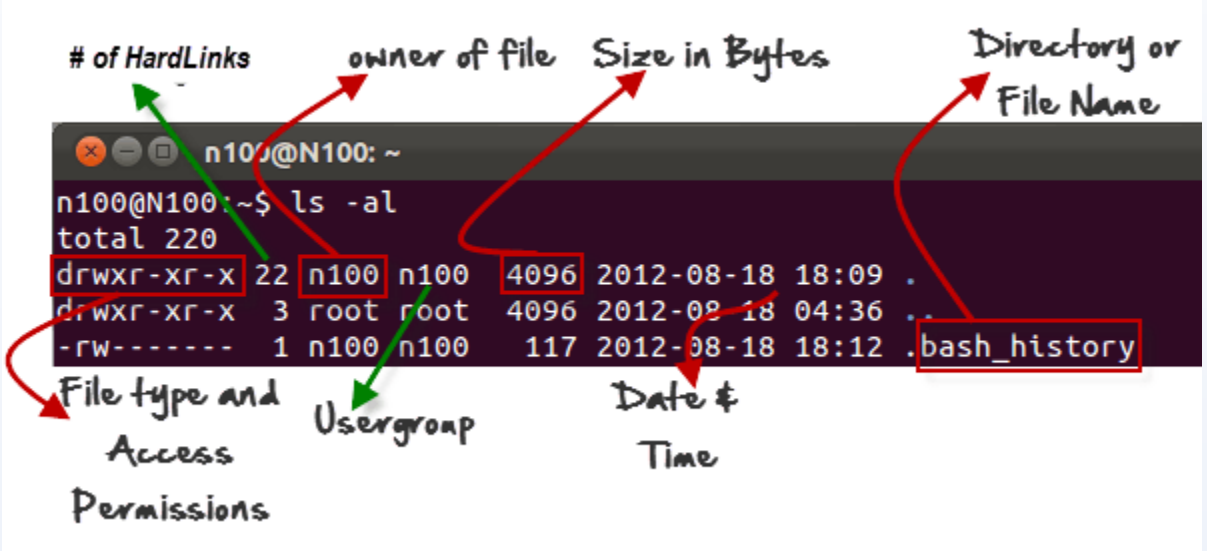
- Các function trong C library: C start-up routine, exit function, exit handler, standard I/O cleanup
- Các syscall: exec, _exit, _Exit
- Các function trong user program: main function, user functions

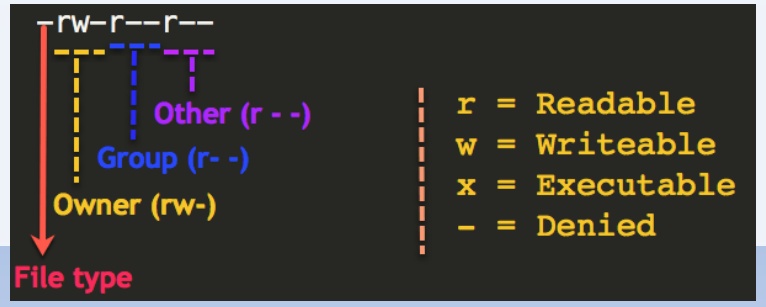
File Permission

Khái niệm permission

- Mỗi file trong hệ thống được gán 9-bit permission để kiểm soát quyền truy cập file
- Hệ thống chỉ cho phép một process truy cập một file nếu user thực thi process đó có đủ quyền
- User được kiểm tra lần lượt theo các nhóm: root user, owner user, group user, other user và sử dụng quyền của nhóm trùng khớp đầu tiên
- Lưu ý: Directory cũng là file trong Linux

Kiểm tra và thay đổi permission





Permission của file được:

- Kiểm tra bằng command *ls -al <path>*
- Thay đổi bằng command *chmod <mode> <path>*

Tác dụng của permission

Permission	File	Directory
read	đọc file	đọc nội dung dir
write	ghi file	-
execute	thực thi file	truy cập file có dir trong file path
write + execute	ghi và thực thi file	tạo file mới trong dir xóa file trong dir

File

open

```
#include <fcntl.h>

int open(const char *path, int oflag, ... /* mode_t mode */ );

int openat(int fd, const char *path, int oflag, ... /* mode_t mode */ );
```

Both return: file descriptor if OK, –1 on error

open(): Mở file tại đường dẫn <path> với tính chất <oflag> (và với mode <mode>)

openat(): Mở file tại đường dẫn <path> so với directory <fd> với tính chất <oflag> (và với mode <mode>)

open <path> và <fd>

open()/openat() xác định path của file cần mở theo bảng bên dưới, trong đó:

- cwd là thư mục làm việc hiện tại
- dir là thư mục được chỉ ra bởi <fd>
- AT_FDCWD là giá trị đặc biệt được định nghĩa trong C library

<path>	open	openat	
		<fd> != AT_FDCWD	<fd> = AT_FDCWD
Tương đối	cwd + <path>	dir+ <path>	cwd + <path>
Tuyệt đối	<path>	<path>	<path>

open <oflag>

C library định nghĩa các constant có thể kết hợp bằng phép logic OR để chỉ ra <oflag>

Bắt buộc chọn 1 trong 5 constant bên dưới	
O_RDONLY	Mở chỉ để đọc
O_WRONLY	Mở chỉ để ghi
O_RDWR	Mở để đọc và ghi
O_EXEC	Mở chỉ để thực thi
O_SEARCH	Mở chỉ để tìm kiếm (dùng cho thư mục)

Các constant tùy chọn thường dùng	
O_APPEND	Ghi vào cuối file
O_CREAT	Tạo một file nếu nó chưa tồn tại. Cần chỉ ra <mode>
O_NOFOLLOW	Tạo ra lỗi nếu <path> là symbolic link
O_SYNC	System call write sẽ chờ đến khi ghi xong vào phần cứng

open <mode>

<mode> chỉ được dùng khi tạo file mới. Các constant bên dưới được kết hợp bằng logic OR để chỉ ra <mode>

S_IRUSR	user-read
S_IWUSR	user-write
S_IXUSR	user-execute
S_IRGRP	group-read
S_IWGRP	group-write
S_IXGRP	group-execute
S_IROTH	other-read
S_IWOTH	other-write
S_IXOTH	other-execute

close

```
#include <unistd.h>

int close(int fd);
```

Returns: 0 if OK, –1 on error

close(): đóng file <fd>

Khi process terminate, tất cả các file đang mở sẽ tự động đóng bởi kernel

lseek

```
#include <unistd.h>
off_t lseek(int fd, off_t offset, int whence);
                                    Returns: new file offset if OK, –1 on error
```

lseek(): thay đổi vị trí đang đọc hoặc ghi của file <fd>

Tùy vào loại file, một số file không hỗ trợ lseek()

lseek <offset> and <whence>

Vị trí đọc hoặc ghi của file được xác định theo bảng bên dưới, trong đó

- SEEK_SET, SEEK_CUR, SEEK_END: constant định nghĩa trong C library
- begin: bắt đầu file
- current: vị trí đang đọc/ghi hiện tại
- size: kích thước file

<whence>	Vị trí đọc/ghi	Điều kiện <offset>
SEEK_SET	begin + <offset>	>= 0
SEEK_CUR	current + <offset>	-
SEEK_END	size + <offset>	-

read

```
#include <unistd.h>

ssize_t read(int fd, void *buf, size_t nbytes);
```

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

read(): đọc file <fd> vào <buf> với kích thước tối đa <nbytes>

write

```
#include <unistd.h>

ssize_t write(int fd, const void *buf, size_t nbytes);

                                    Returns: number of bytes written if OK, –1 on error
```

write(): ghi <buf> vào file <fd> với kích thước <nbytes>

dup

```
#include <unistd.h>

int dup(int fd);
```

return: new file descriptor if OK, –1 on error

dup(): tạo ra file descriptor mới từ <fd>

file descriptor mới luôn nhận giá trị nhỏ nhất chưa được sử dụng

stat

```
#include <sys/stat.h>

int stat(const char *restrict pathname, struct stat *restrict buf);
                                                    return: 0 if OK, –1 on error
```

stat(): đọc thông tin của file <pathname> vào <buf>

stat <buf>

<buf> là con trỏ trỏ tới *struct stat*, loại của file có thể được xác định dựa vào *st_mode* bằng cách sử dụng các macro

Ví dụ: if(S_ISREG(buf->st_mode)) {printf("regular file\n");}

```
struct stat {
  mode_t          st_mode;    /* file type & mode (permissions) */
  ino_t           st_ino;     /* i-node number (serial number) */
  dev_t           st_dev;     /* device number (file system) */
  dev_t           st_rdev;    /* device number for special files */
  nlink_t         st_nlink;   /* number of links */
  uid_t           st_uid;     /* user ID of owner */
  gid_t           st_gid;     /* group ID of owner */
  off_t           st_size;    /* size in bytes, for regular files */
  struct timespec st_atim;    /* time of last access */
  struct timespec st_mtim;    /* time of last modification */
  struct timespec st_ctim;    /* time of last file status change */
  blksize_t       st_blksize; /* best I/O block size */
  blkcnt_t        st_blocks;  /* number of disk blocks allocated */
};
```

Macro	Type of file
S_ISREG()	regular file
S_ISDIR()	directory file
S_ISCHR()	character special file
S_ISBLK()	block special file
S_ISFIFO()	pipe or FIFO
S_ISLNK()	symbolic link
S_ISSOCK()	socket

directory

```
#include <dirent.h>
DIR *opendir(const char *pathname);
                                        return: pointer if OK, NULL on error
struct dirent *readdir(DIR *dp);
                        Returns: pointer if OK, NULL at end of directory or error
int closedir(DIR *dp);
                                        Returns: 0 if OK, –1 on error
```

opendir(): mở dir <pathname>

readdir(): đọc từng file trong dir <dp>

closedir(): đóng dir <dp>

struct dirent: struct chứa thông tin của file, trong đó có chuỗi kí tự *d_name*

Process

fork

```
#include <unistd.h>

pid_t fork(void);
```

Returns: 0 in child, process ID of child in parent, –1 on error

fork(): tạo ra process mới (child process) giống process đang chạy (parent process)

wait

```
#include <sys/wait.h>

pid_t wait(int *statloc);
```

Both return: process ID if OK, 0 (see later), or –1 on error

- wait(): chờ đến khi child process kết thúc, và gán giá trị trả về của child process vào <statloc> nếu <statloc> khác NULL

pipe

```
#include <unistd.h>
int pipe(int fd[2]);
```

Returns: 0 if OK, –1 on error

pipe(): tạo ra 1 pipe cho process và gán file descriptor đọc/ghi pipe vào <fd[0]>/<fd[1]> tương ứng

pipe được sử dụng để trao đổi dữ liệu giữa 2 process có mối liên hệ (ví dụ parrent process và child process)

execv

```
#include <unistd.h>
int execv(const char *pathname, char *const argv[]);
                                    return: –1 on error, no return on success
```

execv(): load và thực thi program <pathname> với tham số <argv> thay thế cho process đang chạy

<argv> là mảng của các con trỏ trỏ tới các tham số theo kiểu chuỗi kí tự

<argv> sử dụng con trỏ NULL để dánh dấu kết thúc mảng

others

getpid()

_exit()

_Exit()

sleep()

link()

unlink()

Exercise

Bài tập

1. Header <errno.h> có công dụng gì? Hàm strerror() và perror() có chức năng gì? Viết (một vài) chương trình minh họa

Lưu ý: các bài tập lập trình phải xử lý lỗi trả về từ các C library function

Bài tập

2. Viết chương trình **mycat** in ra stdout nội dung của các file có đường dẫn là các tham số truyền vào, nếu không nhận được tham số nào thì đọc từ stdin

Ví dụ:

$./mycat in1.txt in2.txt

$./mycat

Bài tập

3. Viết chương trình **myxsfile** nhận tham số là đường dẫn thư mục và in ra stdout <file path: file size> của regular file có kích thước lớn nhất trong thư mục đó và các thư mục con

Gợi ý:

Nên sử dụng đệ quy và bỏ qua file . và .. trong mỗi thư mục

Ví dụ:

$./myxsfile /home/ubuntu

/home/ubuntu/bigdir/bigfile/thebiggest: 45000 byte

Bài tập

4. Viết chương trình **myshell** có những tính năng sau:
    a) Thực thi program(s) có đường dẫn trong thư mục /usr/bin
    b) Redirect output (>) và input (<) tới file
    c) Pipeline tối đa 2 program

Ví dụ:

$./myshell

>> ls ~

>> cat in1.txt in2.txt in3.txt > out.txt

>> cat < in.txt | grep “a” > out.txt