SYSTEM SOFTWARE INTRODUCTION

Nguyen Huu Duc

School of Information and Communication Technology
Hanoi University of Science and Technology

Content

1. Introduction to the course
2. Overview of system software
3. Syllabus

Lecturer

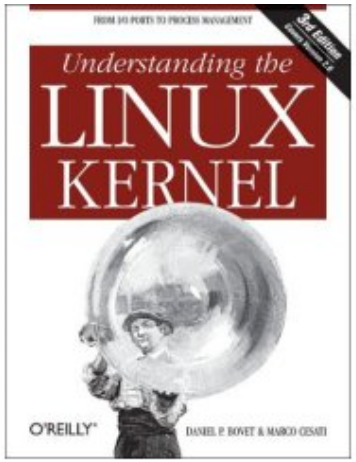
- Full name: NGUYEN HUU DUC
- Affiliation (1): Department of Information System, SoICT, HUST
- Affiliation (2): Center for Data and Computation Technologies, HUST
- Email: duc.nguyenhuu@hust.edu.vn
- Tel.: (84) 24 38696124
- Mobile: 0975651915
- Homepage: `http://hpcc.hut.edu.vn/~ducnh`

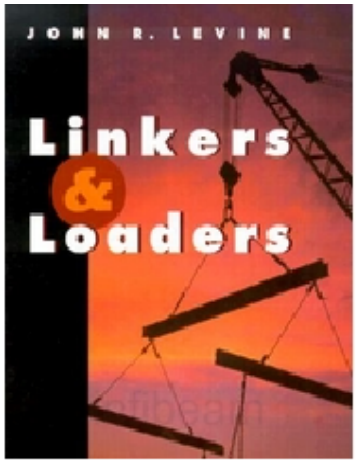
Course Information

- Code: 08-06 (1-1)
- Quantity: 30 theoretic sessions
- Description:
  - System software is computer software designed to operate the computer hardware and to provide a platform for running application software.
  - E.g. Compiler, Linker, Assembler, Library, Operating system, etc.
- Objectives:
  - Master the role, function and structure of system softwares
  - Understand relationships between system softwares.
  - Master the structure of an application software, and how to implement those application softwares.
  - Know how to develop your own system software.

Reference materials

- Daniel P. Bovet, Marco Cesati, Understanding the Linux Kernel, O'Reilly, 2nd edition, 2002
- John R. Levine, Linkers and Loaders, Morgan Kaufmann Publishers, 2000
- Jonathan Corbet, Alessandro Rubini, Greg Kroah-Hartman, Linux Device Drivers, 3rd edition, O'Reilly, 2005





Requirements for student

- You must strictly follow the following things:
  - Full participation is fully expected of every student in the class as provided in the bylaws
  - Completing every assignment in the class.
  - Doing your long-term assignment.
- Grade
  - Long-term assignment (30%)
    - Building a system software (System call/Device driver).
  - Final examination (70%)

What is a system software?

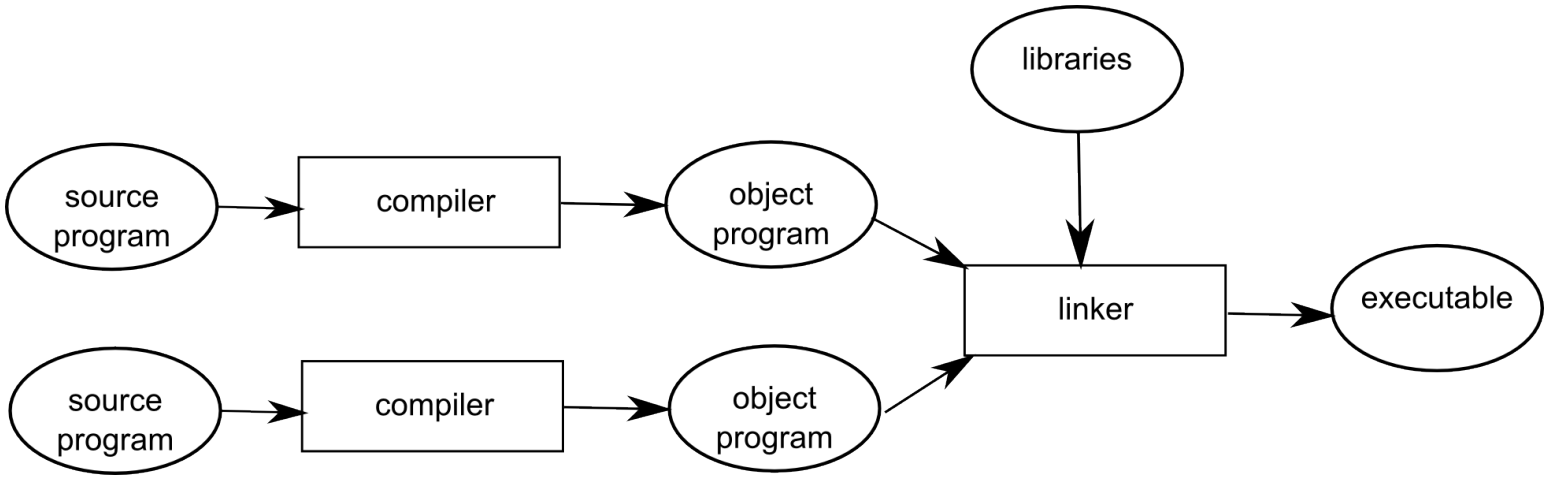
Definition (Computing Dictionary)

- Is any software required to support the production or execution of application programs but which is not specific to any particular application.
- System software typically includes an operating system to control the execution of other programs; user environment software such as a command-line interpreter, window system, desktop; development tools for building other programs such as assemblers, compilers, linkers, libraries, interpreters, cross-reference generators, version control, make; debugging, profiling and monitoring tools; utility programs, e.g. for sorting, printing, and editing.

Why should we learn about system software?

- To bridge the gap of knowledge between application software and computer hardware.
  - To understand how application softwares are created.
- To have knowledge to build a specific system software (E.g. device driver).

Build a application software



Example

- Compile

  ```
  #gcc -c a.c b.c
  ```

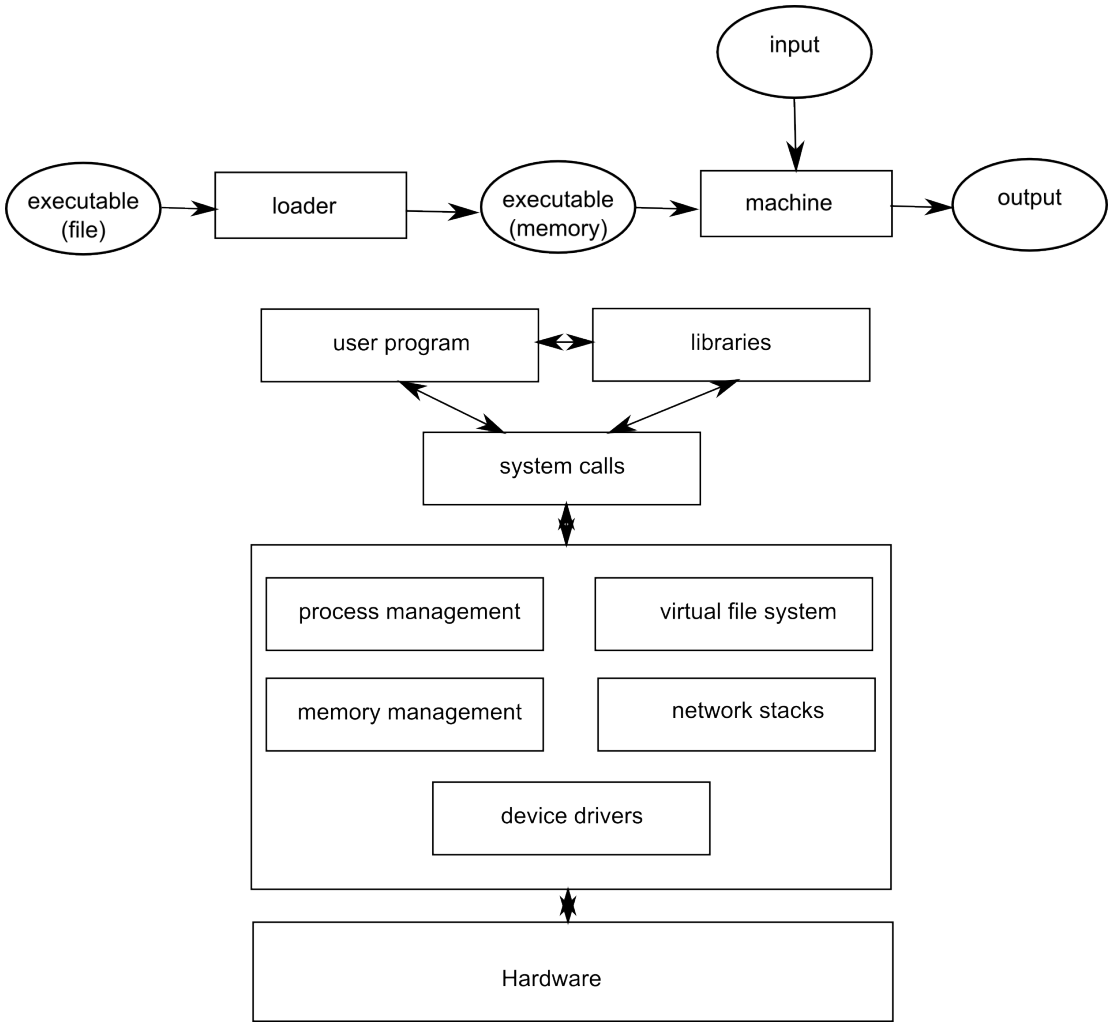
- Linking

  ```
  #gcc -o myprog a.o b.o
  ```

- Load and execute

  ```
  #./myprog
  ```

Execute a system software



Source program

- An original computer program written by a programmer in the plain text format.
- Includes one or more files.
- Written in the syntax of one or some programming languages.
  - High-level programming language: C, JAVA
  - Low-level programming language: Assembly
- Be edited by tools such as: Text editor or Integrated Development Environment.
- Be compiled into object code by compiler, or executed on computer using interpreter.

Example

Example

```
#include <stdio.h>

int main(void)
{
  printf("Hello World!\n");
  return 0;
}
```

- Not the program that can be executable
  - Need to be compiled and linked to get executable code.
- Can not be loaded into memory for execution
- Not include the declaration of function `printf`
  - Being in library (E.g. `libc`)

Object program

- Is the output of compilation of a source program.
- It is usually represented in machine language (Be compatible with a specific processor)
  - Intel x86/IA32
  - MIPS
- Some addresses referring to code snippets or data area may be lacked due to separate compilation.
  - E.g. Reference to the function `printf` in the library
- Can not be loaded into memory and executed

Example

Example

```
00000000 <main>:
   0:   8d 4c 24 04             lea    0x4(%esp),%ecx
   4:   83 e4 f0                and    $0xfffffff0,%esp
   7:   ff 71 fc                pushl  0xfffffffc(%ecx)
   a:   55                      push   %ebp
   b:   89 e5                   mov    %esp,%ebp
   d:   51                      push   %ecx
   e:   83 ec 04                sub    $0x4,%esp
  11:   c7 04 24 00 00 00 00    movl   $0x0,(%esp)
  18:   e8 fc ff ff ff          call   19 <main+0x19>
  1d:   b8 00 00 00 00          mov    $0x0,%eax
  22:   83 c4 04                add    $0x4,%esp
  25:   59                      pop    %ecx
  26:   5d                      pop    %ebp
  27:   8d 61 fc                lea    0xfffffffc(%ecx),%esp
  2a:   c3                      ret
```

Library

- Includes a collection of sub-programs or classes that are used to develop source program.
- You can consider it a set of object files which were developed and compiled.
- Programmer can collect, archive, and re-use sub-programs (classes) in the library.
  - Definition of function `printf` is in the library `libc`
- Types of library
  - Static linking library.
  - Dynamic linking library.

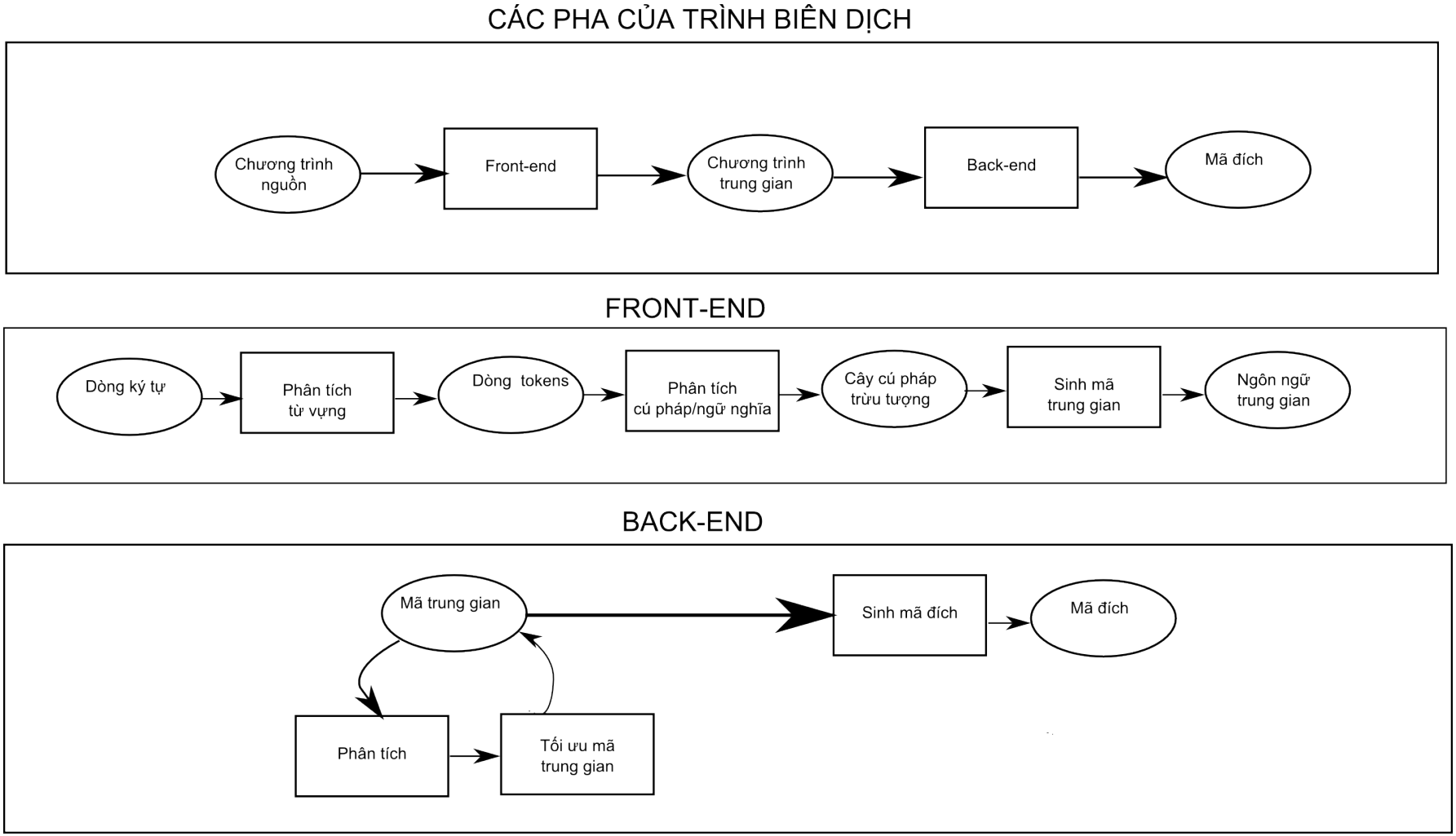
Executable code

- Is a program represented by machine language and ready for execution.
- Is usually stored in storage device (E.g. Hard drive) and loaded into main memory at the time of execution.
- Is usually the result of two phases: compilation and linking.

Compiler

- Is a computer program which plays a role of converting source programs (written in high-level programming language) to target programs (written in low-level programming language) with conservation of semantics
- E.g. `gcc`, `javac`

Compiler structure



Assembler

- An assembly language is a low-level programming language for microprocessors, microcontrollers and other programable devices.
  - It implements a symbolic representation of the machine codes and other constants needed to program a given CPU architecture.
  - This representation is usually defined by the hardware manufacturer
  -
- An assembler is used to translate assembly language statements into the object file.
  - Translate mnemonics into machine instructions (opcodes).
  - Translate names into memory address.

Linker

- Linker is a computer program that takes one or more objects generated by a compiler and combines them into a single executable program
- The linker takes care of arranging the objects in a program's address space.
- Relocating machine code may involve re-targeting of absolute jumps, loads and stores.

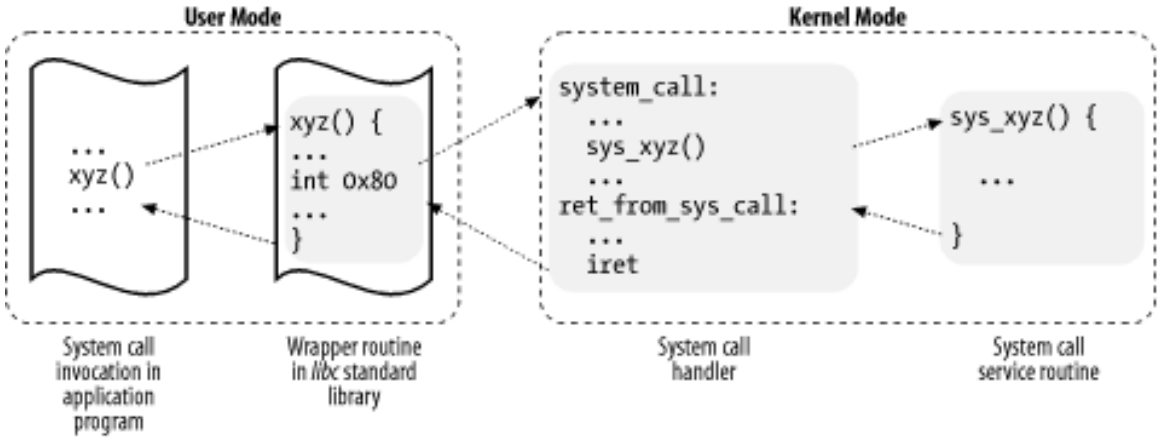
Operating system

- An operating system is a set of programs that manage computer hardware resources and provide common services for application software.
  - Manage computer hardware resources
    - CPU
    - Memory
    - File system
    - Network devices
    - Peripheral device (Monitor, keyboard, mouse, etc.)
  - Provide service
    - Program execution service (System call, device driver, etc.)
    - User service (User management, user interface, etc.)

System calls

- System calls is services provided by operating system, it executes a specific set of instructions over which the calling program has no direct control.
  - Process Control (create process, load, execute, terminate, etc.)
  - File management (create file, delete file, open, close, read, write, etc.)
  - Device Management (request device, release device, read, write, get/set device attributes, etc.)
  - Information maintenance (get/set time or date, get/set system data, etc.)
  - Communication (create, delete communication connection, send, receive messages etc.)
- System call interface is the method to execute system call from application program.
  - Using interrupt

System calls



Device drivers

- A device driver is a computer program allowing higher-level computer programs to interact with a hardware device.
  - A calling program requests device functions through a call to a device driver.
  - The device driver extracts requests into commands sending to a device.
  - The device sends data back to the driver through interrupt mechanism.
  - The driver may invoke routines in the original calling program
- Linux device drivers
  - In Linux, a device driver is a module that is linked with operating system kernel.
  - Provides services supporting communication with device.
    - Character device driver
    - Block device driver
    - Network device driver

Archiver

- An archiver is a computer program that combines a number of files together into one archive file, or a series of archive files, for easier transportation or storage.
  - Create an archive file.
  - Modify an archive file (add, delete components).
  - Extract data from an archive file.
- Is used to manage sub-programs in a library.

Syllabus

Part I. Introduction (1 lesson)

Part II. User programs (3 lessons)

Part III. Object files (2 lessons)

Part IV. Linker (3 lessons)

Part V. Startup process (1 lesson)

Part VI. System calls (3 lessons)

Part VII. Device Driver (2 lessons)