CS343: Operating System

OS Top-down Approach and System Program

Lect07 : 11th Aug 2023

Dr. A. Sahu
Dept of Comp. Sc. & Engg.
Indian Institute of Technology Guwahati

Outline

- OS : top down approaches
- System Program
- Static Linking and Dynamic Linking
- OS Structures
  - How the OSs are structured/organized
- Different type of OS
  - Desktop, Android, Cloud, Peers, Embedded, RealTime

OS Management: Top Down Approach

- Process
- Memory
- Storage
- I/Os Subsystem
- Protection and Security

So user need **system call service of OS** for all above items

Storage and I/O Management
&
Related System Call

Storage Management

- OS provides uniform, logical view of information storage
  - Abstracts physical properties to logical storage unit - **file**
  - Each medium is controlled by device (i.e., disk drive, tape drive)
    - Varying properties include access speed, capacity, data-transfer rate, access method (sequential or random)
- File System Management

File System Management

- Files usually organized into directories
- Access control on most systems to determine who can access what
- OS activities include
  - Creating and deleting files and directories
  - Primitives to manipulate files and directories
  - Mapping files onto secondary storage
  - Backup files onto stable (non-volatile) storage media

I/O Subsystem

- **One purpose of OS is to hide peculiarities of hardware devices from the user**
- I/O subsystem responsible for Mem. Mngt of I/O
  - Buffering (storing data temporarily while it is being transferred),
  - Caching (storing parts of data in faster storage for performance),
  - Spooling (the overlapping of output of one job with input of other jobs)
- General device-driver interface
- Drivers for specific hardware devices

File & Device : System Calls

- **File management**
  - create file, delete file, open, close file
  - read, write, reposition
  - get and set file attributes
- **Device management**
  - request device, release device
  - read, write, reposition
  - get device attributes, set device attributes
  - logically attach or detach devices

Communication: System Calls

- **Communications**
  - create, delete communication connection
  - send, receive messages if **message passing model** to **host name** or **process name:** From **client** to **server**
  - **Shared-memory model** create and gain access to memory regions
  - transfer status information
  - attach and detach remote devices

Protection and Security

- **Protection** – any mechanism for controlling access of processes or users to resources defined by the OS
- **Security** – defense of the system against internal and external attacks
  - Huge range, including denial-of-service, worms, viruses, identity theft, theft of service
  - **Privilege escalation** allows user to change to effective ID with more rights

Protection and Security

- Systems generally first distinguish among users, to determine who can do what
  - User identities (**user IDs**, security IDs) include name and associated number, one per user
  - User ID then associated with all files, processes of that user to determine access control
  - Group identifier (**group ID**) allows set of users to be defined and controls managed, then also associated with each process, file
  - **Privilege escalation** allows user to change to effective ID with more rights **Ex: sudo**

Protection & Info. Maintenance: System Calls

- **Information maintenance**
  - get time or date, set time or date
  - get system data, set system data
  - get and set process, file, or device attributes
- **Protection**
  - Control access to resources
  - Get and set permissions
  - Allow and deny user access

Windows and Unix System Calls

	Window	Linux
Process Control	CreatePrtocess()	fork()
	Exit Process()	exit()
	WaitForSingleObject()	wrait()
File Manipulation	CreateFile()	open()
	ReadFile	read()
	WriteFile()	write()
	CloseHandle()	close()
Device Manipulation	SetControlMode()	ioctl()
	ReadConsole()	read()
	WriteConsole()	write()

Windows and Unix System Calls

	Window	Linux
Information Maintenance	GetCurrentProcessID()	getpid()
	SetTimer()	alarm()
	Sleep()	sleep()
Commu-nication	CreatePipe()	pipe();
	CreateFileMapping()	shmget()
	MapViewOfFile()	mmap()
Protection	SetFileSecurity()	chmod()
	InitializeSecurityDescriptor()	umask()
	SetSecurityDescriptor Group()	chown()

System Program

System Programs: not the OS

- System programs
  - Provide a convenient ENVT for **program development** and **execution**.
- **Confusing: with OS definition**
- OS Provide a convenient ENVT
  - To **Execute user programs** and make solving user problems easier
  - Acts as an intermediary between a user of a computer and the computer hardware
  - Make the computer system convenient to use

Examples: System Program

- Linker, Assembler, Loader, Compiler
- IDE: Kdevelop, TC++, DevC++, VisualC++
- Compiler: GCC, ICC, JavaC, JDK, NVCC
- Assembler : NASM, MASM
- Loader : ld command
- Debugger : GDB
- **These are not System Program**
  - Microsoft Word, PhotoShop, ImageMagic, VLC Player, Firefox, CarRace

Are library files system program ?

- **Tools**
  - Compiler, Linker, Loader, Simple Text Editor (vim,gedit)
- Almost all standard library are system program
  - As they are part of compiler/tool
- User created library are not system program
- All library may use the system call or high level API

System Programs: not the OS

- System Program can be divided into:
    - File manipulation
    - Status info. : sometimes stored in a File
    - Programming language support
    - Program loading and execution
    - Communications
    - Background services
- Most users' view of the **operation system** is defined by **system programs**, not the actual **system calls**

System Programs

- **File management**
  - Create, delete, copy, rename, print, dump, list, and generally manipulate files and directories
- **File modification**
  - Text editors to create and modify files
  - Special commands to search contents of files or perform transformations of the text

System Programs

- **Status information:**
  - Some ask the system for information
    - Date, Time
    - Amount of available memory, disk space
    - Number of users
    - Others provide detailed performance, logging, and debugging information
  - Typically, these programs format and print the output to the terminal or other output devices
  - Some systems implement a **registry** - used to store and retrieve configuration information

System Programs (Cont.)

- **Programming-language support** -
  - Compilers , Assemblers, Debuggers and interpreters
- **Program loading and execution**-
  - Absolute loaders, Relocatable loaders,
  - linkage editors, and overlay-loaders,
  - Debugging systems for higher-level and machine language
- **Communications** – Provide mechanism for
  - Creating virtual connections among processes, users, and computer systems

System Programs (Cont.)

- **Background Services**
  - Launch at boot time
    - Some for system startup, then terminate
    - Some from system boot to shutdown
  - Provide facilities like disk checking, process scheduling, error logging, printing
  - Run in user context not kernel context
  - Known as **services**, **subsystems**, **daemons**

Application Program

- **Application programs**
  - Don’t pertain to system, Run by users
  - Not typically considered part of OS
  - Launched by command line, mouse click, finger poke

Fun time

Static Linking
and
Dynamic Linking

Compiling multiple Files

```
//foo.c
int foo3x(int x){
        return 3*x;
 }
```

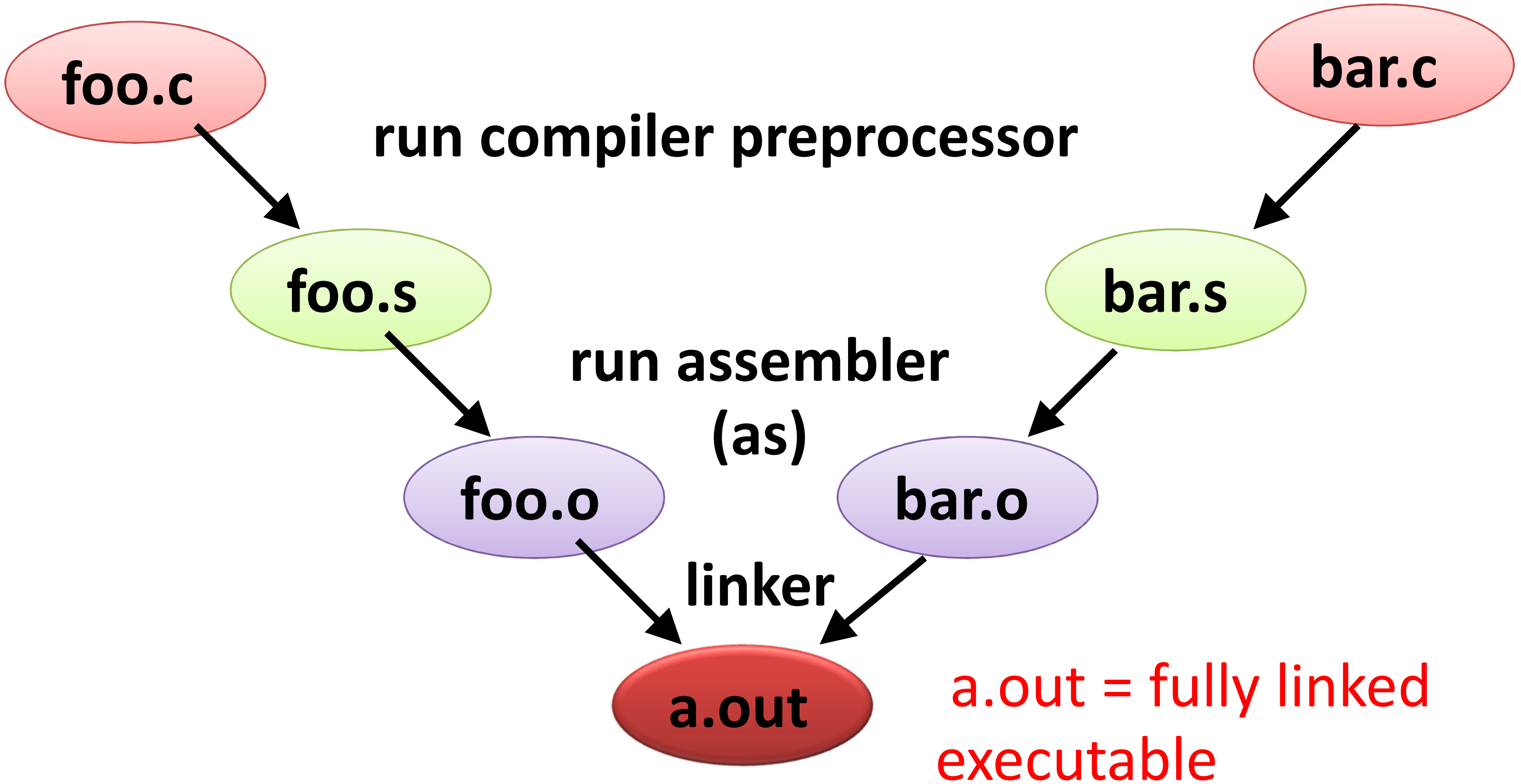
```
//bar.c
int main(){
        int x;
        x=foo3x(10);
        printf("%d",x);
        return  0;
 }
```

- $ gcc –c foo.c
- $ gcc –c bar.c
- $ gcc foo.o bar.o
- $./a.out

Linker and Loader

- Compiler in Action…

```
$gcc   foo.c   bar.c   –o a.out
```



What is Linker ?

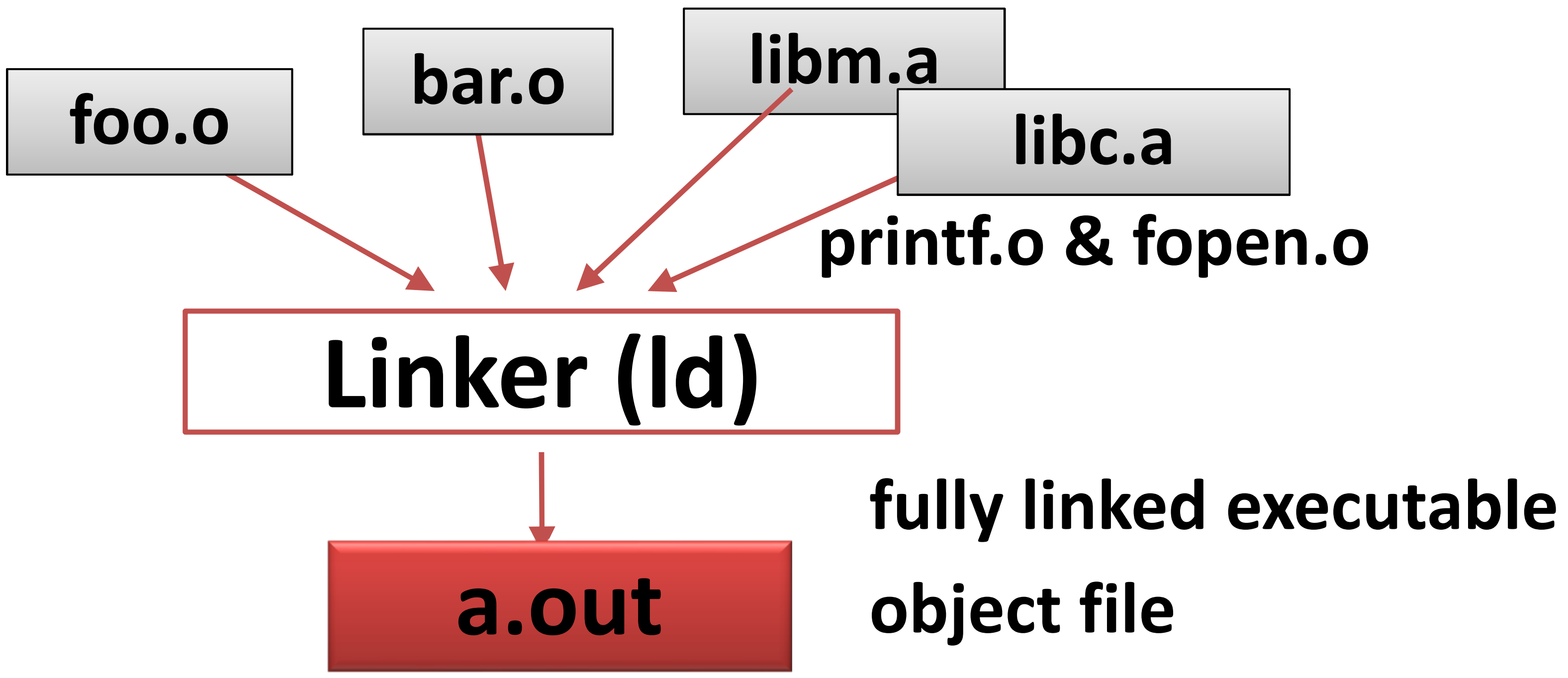
- Combines multiple relocatable object files
- Produces fully linked executable – directly loadable in memory
- How?
  - Symbol resolution – associating one symbol definition with each symbol reference
  - Relocation – relocating different sections of input relocatable files

Object files

- Types –
  - Relocatable : Requires linking to create executable
  - Executable : Loaded directly into memory for execution
  - Shared Objects : Linked dynamically, at run time or load time

Linking with Static Libraries

- Collection of concatenated object files – stored on disk in a particular format – archive
- An input to Linker
  - Referenced object files copied to executable



Creating Static Library

```
//foo.c
int foo3x(int x){
        return 3*x;
 }
```

```
int main(){//bar.c
        int x;
        x=foo3x(10);
        printf(“%d”,x);
        return  0;
 }
```

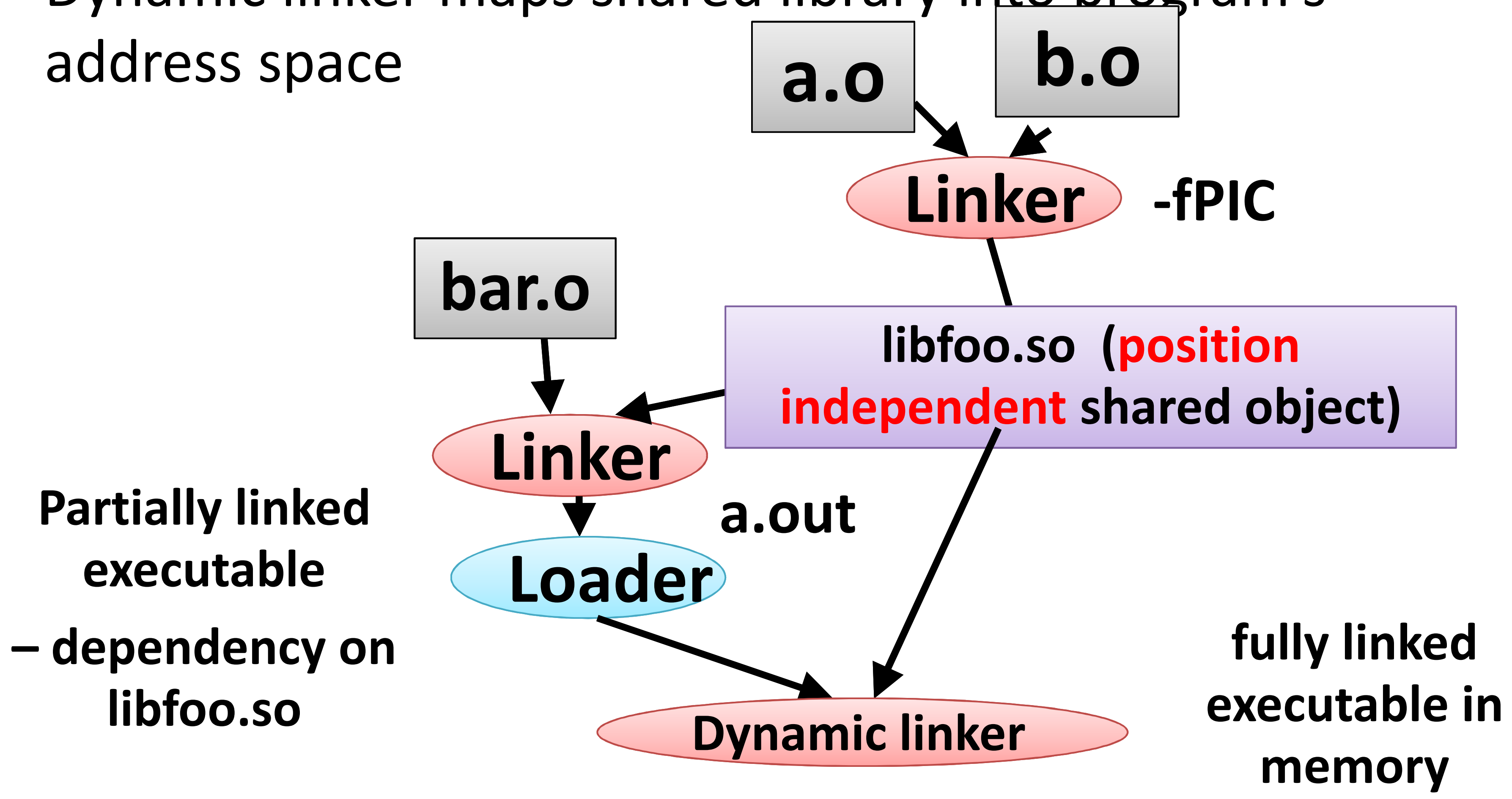
- **$ gcc -c foo.c**
- **$ ar rcs libfoo.a foo.o   //it create libfoo.a**
- **$ gcc bar.c -L. -lfoo**
- **$ ./a.out**

Dynamic Linking – Shared Libraries

- Addresses disadvantages of static libraries
  - Ensures one copy of text & data in memory
  - Change in shared library does not require executable to be built again
  - Loaded at run-time by dynamic linker, at arbitrary memory address, linked with programs in memory
  - On loading, dynamic linker relocates text & data of shared object

Shared Libraries ..(Cntd)

- Linker creates **libfoo.so** (PIC) from **a.o** and **b.o**
- **a.out** – partially executable – depend on **libfoo.so**
- Dynamic linker maps shared library into program's address space


a.o
b.o
Linker
-fPIC
bar.o
libfoo.so (position independent shared object)
Linker
a.out
Loader
Partially linked executable
– dependency on libfoo.so
Dynamic linker
fully linked executable in memory

Creating Dynamic Library

```
//foo.c
int foo3x(int x){
        return 3*x;
 }
```

```
int main(){//bar.c
        int x;
        x=foo3x(10);
        printf("%d",x);
        return  0;
 }
```

- $gcc -c -fPIC foo.c
- $gcc -shared -Wl,-soname,libfoo.so.1 -o libfoo.so.1 foo.o
- $ gcc bar.c -L. –lfoo
- $ export LD_LIBRARY_PATH=.
- $ ./a.out