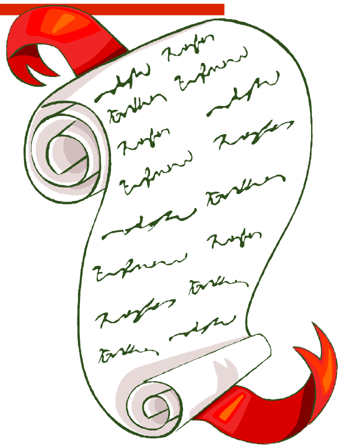
Video Lecture # 02
C-Compilation Process
(System Programmer Perspective)

Course: SYSTEM PROGRAMMING

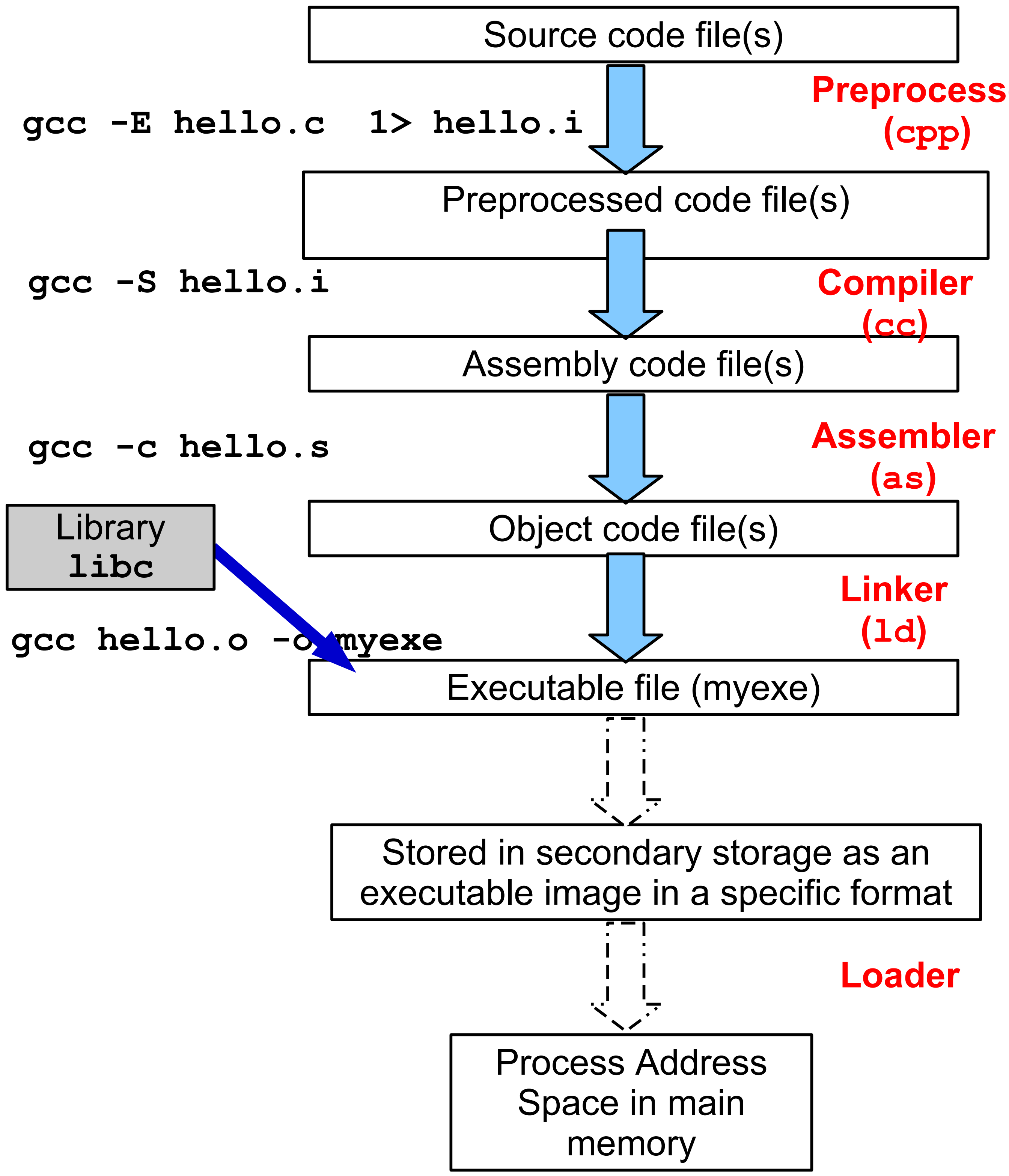
Instructor: Arif Butt

Punjab University College of Information Technology (PUCIT)
University of the Punjab

Today's Agenda



- Review of C Compilation process
- Format of object files
- Viewing the contents of object files
- Loading a program in memory
- Layout of a process in memory
- How to invoke system calls
- How a system call executes



```
gcc -E hello.c  1> hello.i
gcc -S hello.i
gcc -c hello.s
gcc hello.o -o myexe
```

Preprocessor (cpp)

- Interpret preprocessor directives
- Include header files
- Expand macros
- Remove comments

Compiler (cc)

- Checks for syntax errors
- Converts the src to assembly of underlying processor

Assembler (as)

- Generates relocatable object files to be used by linker
- Contains symbol table

Linker (ld)

- Static vs Dynamic linking
- Contains code and data for all functions defined in src files
- Contains global symbol table

Types of Object Files (Modules)

- **Relocatable object file: (.o file)** Contains binary code and data in a form that can be combined with other relocatable object files at compile time to create an executable object file. Each .o file is produced from exactly one .c file. Compilers and assemblers generate relocatable object files.
- **Executable object file: (a.out file)** Contains binary code and data in a form that can be copied directly into memory and executed. Linkers generates executable object files.
- **Shared object file: (.so file)** A special type of relocatable object file that can be loaded into memory and linked dynamically, at either load time or run time. Called dynamic link libraries (dlls) in Windows. Compilers and assemblers generate shared object files.
- **Core file:** A disk file that contains the memory image of the process at the time of its termination. This is generated by system in case of abnormal process termination.

Formats of Object Files (Modules)

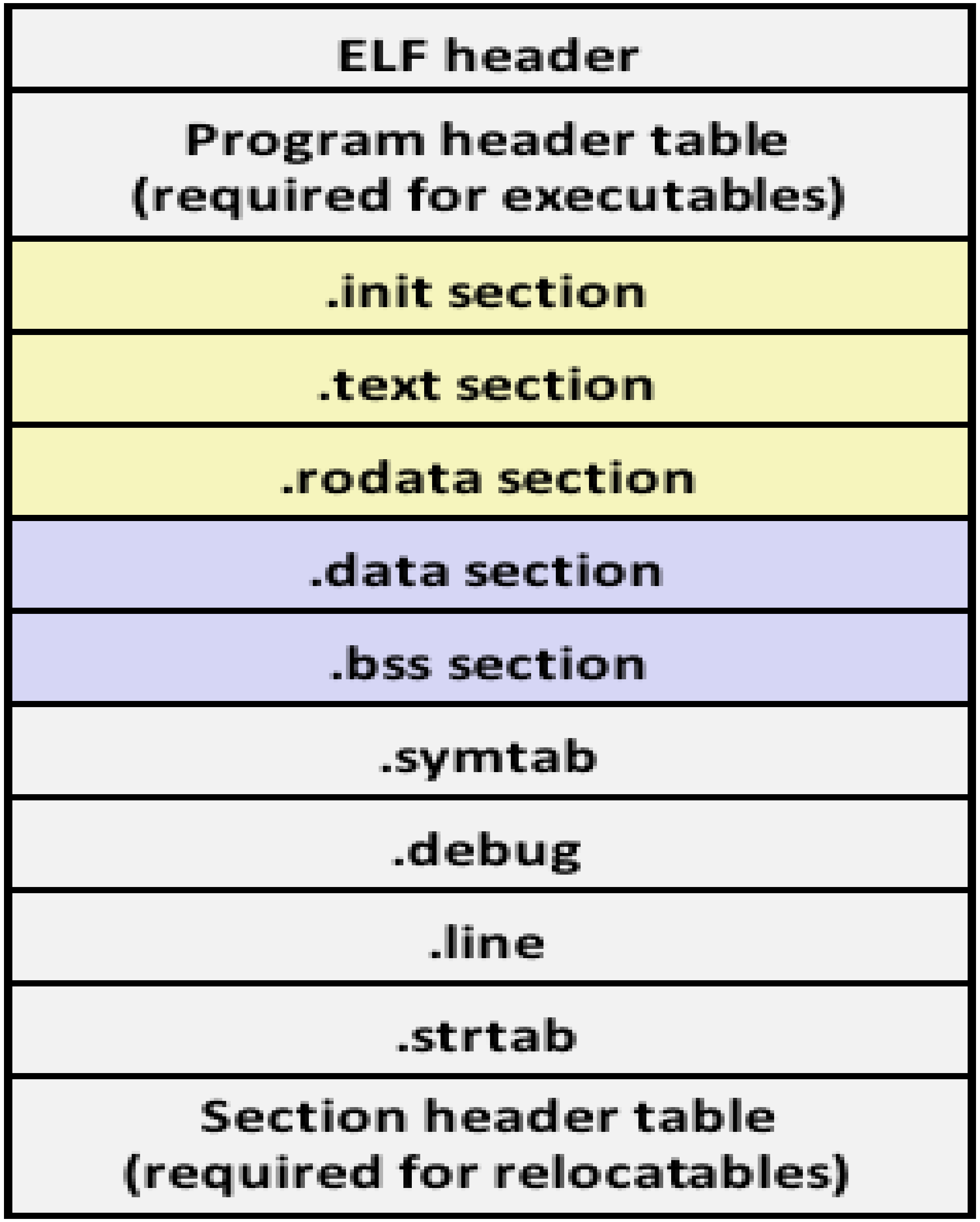
Object file formats vary from system to system. Some famous formats are mentioned below:

Formats	Description
a.out	Original file format for UNIX. It consists of **three** sections: text, data, and bss, which are for program code, initialized data and uninitialized data respectively.
COFF	Common Object File Format was introduced with SVR3 Unix. COFF files may have multiple sections, each prefixed by a header. The number of sections is limited. The COFF specification includes support for debugging but the debugging info was limited. Later ECOFF was introduced by MIPS and XCOFF by IBM
ELF	Executable and Linking Format came with SVR4 UNIX. ELF is similar to COFF in being organized into a number of sections, but it removes many of COFF's limitations. ELF is used on most modern UNIX systems, including GNU/Linux, Solaris and Irix. Also used on many embedded systems
PE	Portable Executable format is used by Windows for their executables. PE is basically COFF with additional headers. The extension normally is .exe

ELF Format

- Executable and Linking Format is binary format, which is used in SVR4 Unix and Linux systems
- It is a format for storing programs or fragments of programs on disk, created as a result of compiling and linking
- ELF not only simplifies the task of making shared libraries, but also enhances dynamic loading of modules at run time
- An executable file using the ELF format consist of ELF Header, Program Header Table and Section Header Table
- The files that are represented in this formats are:
  - Relocatable file objects (**.o**)
  - Normal executable files (**a.out**)
  - Shared object files (**.so**)
  - Core files

ELF Format (cont...)

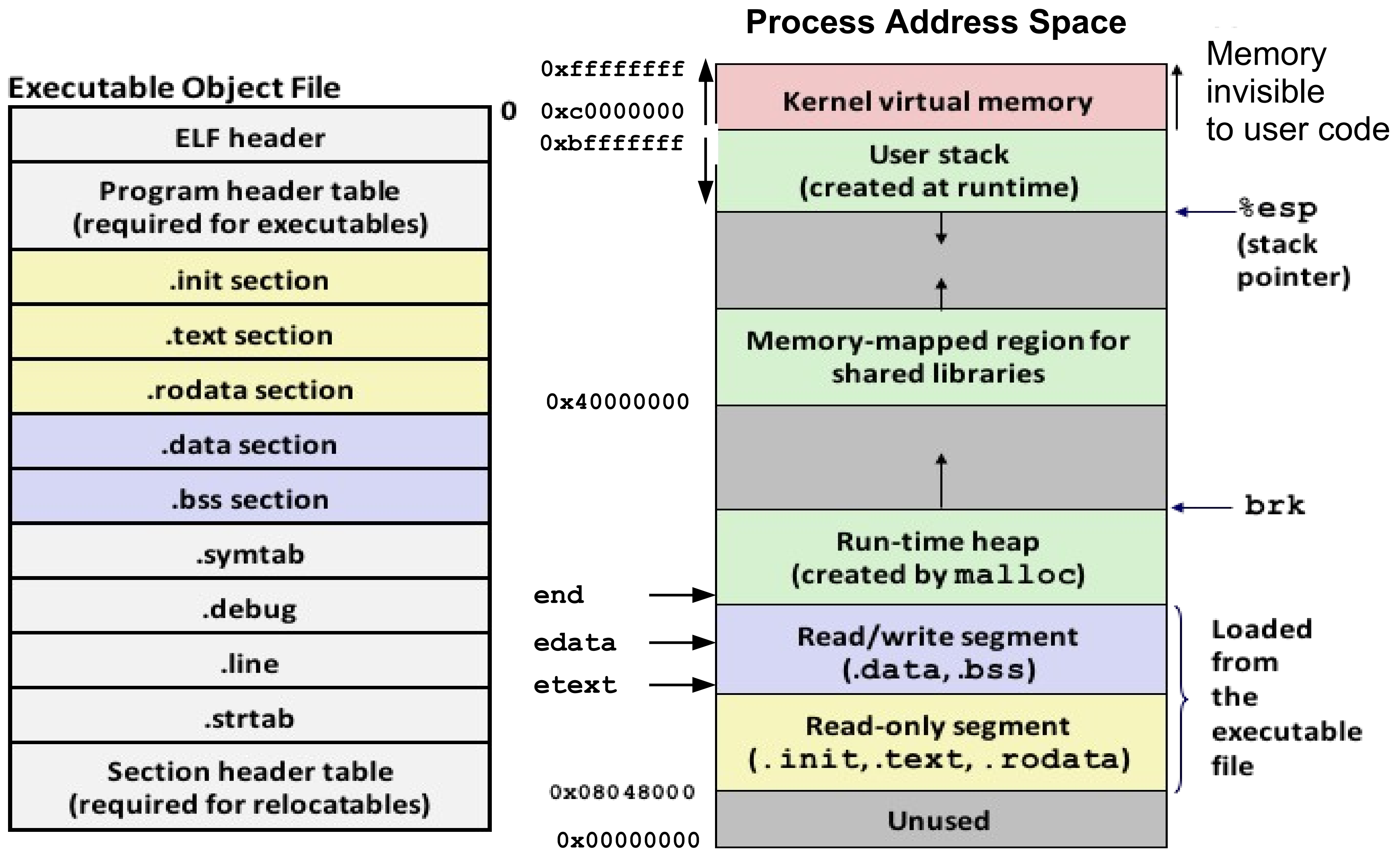


Reading Contents of Object Files
`readelf, objdump`

Loading program in gdb & Viewing CPU Registers

Loading executable in Memory

Loading Executable File in Memory



Startup Routine in `crt1.o`

```
0x08048000 <_start>:  /* entry point in .text  */
call __libc_init_first/* startup code in .text */
call _init            /* startup code in .init */
call atexit            /* startup code in .text */
call main             /* application main func */
call _exit            /* returns control to OS*/
```

Calling a system call using its wrapper

Calling a system call without its wrapper

Calling a system call from Assembly Code

Things To Do

If you have problems visit me in counseling hours. . . .