PURDUE UNIVERSITY®

# CS 25200: Systems Programming

## Lecture 8: Memory Allocation Wrap-up
## Program Generation and Loading

Prof. Turkstra

# Lecture 08

- Memory allocation wrap-up
- Executable formats
- Building a program
- Libraries
- Loader

# Memory smashing

- Program writes to memory outside of the intended data structure or allocated range
- Corrupts the following object(s)
  - In the heap, may overwrite boundary tags or even fencepost(s)
- Sometimes nothing happens
  - Why?

# Example

```
char *s = malloc(8);
strcpy(s, “Hello, World!”);
```

# Debugging memory allocation errors

- Can be very difficult
  - Corruption of malloc's internals may not become apparent until much later
  - Memory leaks may not be immediately obvious
- Trying to find premature/double/wild frees?
  - Can try commenting out all free() calls
  - If the problem goes away, uncomment them one-by-one

# Debugging

- ...or use a tool
- Free
  - Valgrind
  - Memcheck
- Not free
  - IBM Rational Purify
  - Bounds Checker
  - Insure++

# Tradeoffs

- Explicit memory allocation is...
  - Fast
  - Efficient with regard to memory usage
- It also...
  - Can be error prone
  - Requires more expertise
  - May take longer to develop and debug
  - Can be insecure if not done right

- Some prefer to use languages like Java or C#
- C/C++ is still widely used
  - Especially for lower-level, performance critical applications

# Debugging allocators

- Can always augment your allocator to make it more robust
- Check if a block was returned by malloc() using a magic number in the header
  - Prevents wild free()s
- Use different magic numbers to indicate free/allocated
  - Instead of a single bit
  - E.g. free: 0xF7EEF7EE; allocated: 0xA10CA7ED
  - Always check the magic number

- Create an integrity checking function
  - Iterate over all free and allocated blocks
  - Check for two consecutive free blocks
  - Ensure there are no unexpected gaps
- Store size of gaps in the fence posts
- Verify sizes and flags in header match
- Print free and allocated blocks

# Gaps

- Often gaps between segments
- Attempting to access an address in an unmapped region causes the OS to send the process a signal: SIGSEGV
  - Segmentation violation
  - By default program is immediately terminated and dumps core
- This also happens if you access a mapped but protected region
  - E.g., try executing in the data segment

# Core dump file

- $ man 5 core
- File containing an image of the process' memory at time of termination
- Can be used with gdb (or other debuggers)
- May have to enable it (e.g., on data.cs.purdue.edu)
  - $ ulimit -c unlimited

# Program

- File in a particular format containing necessary information to load an application into memory and execute it
  - Often time part of this is split off into the “loader” and libraries
- Programs include:
  - Machine instructions
  - Initialized data
  - List of library dependencies
  - List of memory sections
  - List of values determined at load time

# Executable file formats

- Fully “compiled” programs can come in many different formats…
  - ELF – Executable Linux File
    - Used by most recent UNIX systems (e.g., Solaris, Linux)
  - PE – Portable Executable
    - Used by Windoze
  - Mach-O – Mach object [file format]
    - Used by macOS/iOS
  - COFF – Common Object File Format
    - Also windoze, some embedded systems - historically System V Unix
  - a.out – Used in BSD (Berkeley Standard Distribution)
    - Restrictive, rarely seen anymore

# ELF

- File header
  - Magic number
  - Version
  - Target ABI
  - ISA
  - Entry point
  - Pointers to
    - Program header
    - Section header
  - etc

# Program header

- How to create the process image
  - Segments
  - Types
  - Flags
  - File offset
  - Virtual address
  - Size in file
  - Size in memory

# Section header

- Type (data, string, notes, etc)
- Flags (writable, executable, etc)
- Virtual address
- Offset in file image
- Size
- Alignment

- readelf --headers /bin/ls
- objdump -p, -h, -t

# Purdue "All-American" Marching Band

- Will be performing tomorrow starting at 10:30 AM, Slayter Center

# Building a program

- Start with source code
  - abs.c
- Prepocessor
- Compiler
- Assembler
- Linker

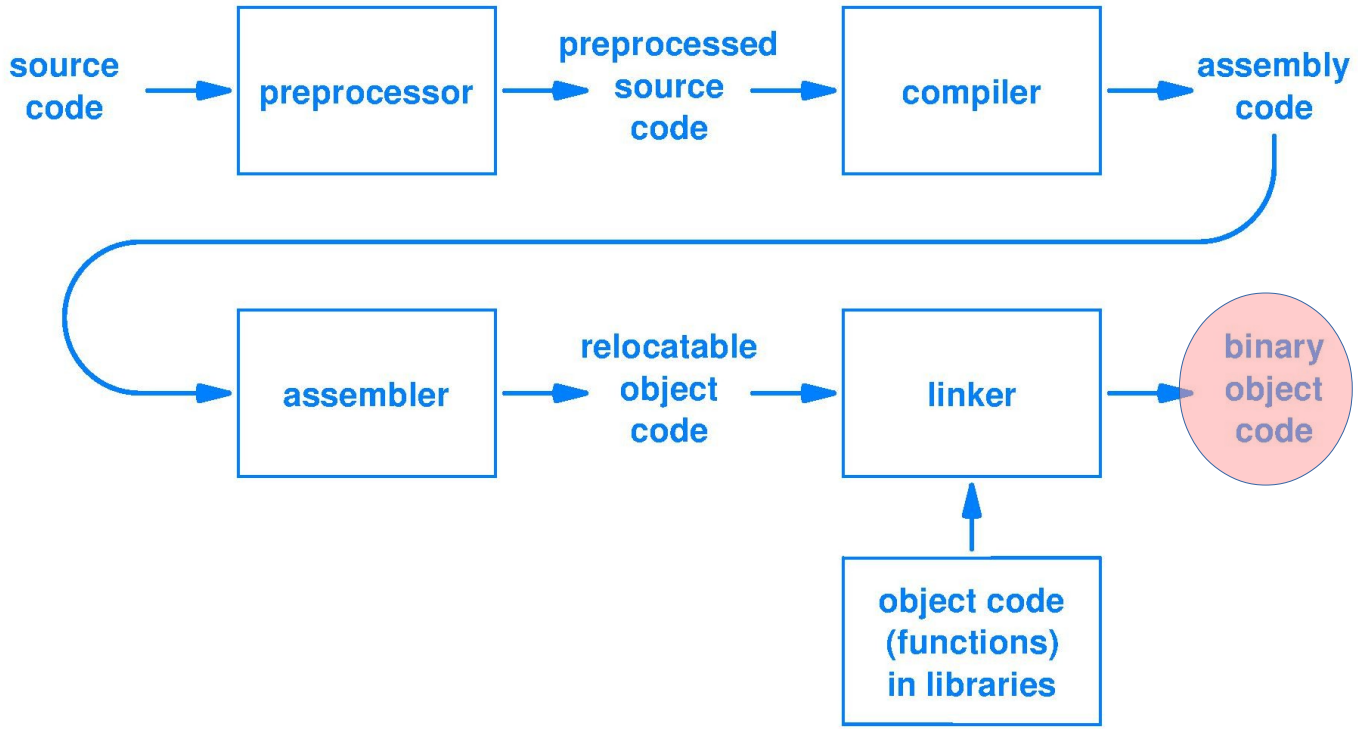



**Figure 4.6** The steps used to translate a source program to the binary object code representation used by a processor.

# Preprocessor

- When a .c file is compiled, it is first scanned and modified by the preprocessor before being handed to the real compiler
- Finds lines beginning with #, hides them from the compiler, or takes some action
- #include, #define
- #ifdef, #else, #endif

# Why macros?

- Run time efficiency
  - No function call overhead
- Passed arguments can be any type
  - #define MAX(x,y) ( (x) > (y) ? (x) : (y) )
  - Works with ints, floats, doubles, even chars

- Can do math
  - #if (FLAG % 4 == 0) || (FLAG == 13)
- Macros
  - #define INC(x)  x+1
  - No semi-colon
  - Have to be careful
    - #define ABS(x)  x < 0 ? -x : x
    - ABS(B+C)

- Parentheses around substitution variables

```
#define ABS(x)  ( (x) < 0 ? -(x) : (x) )
```

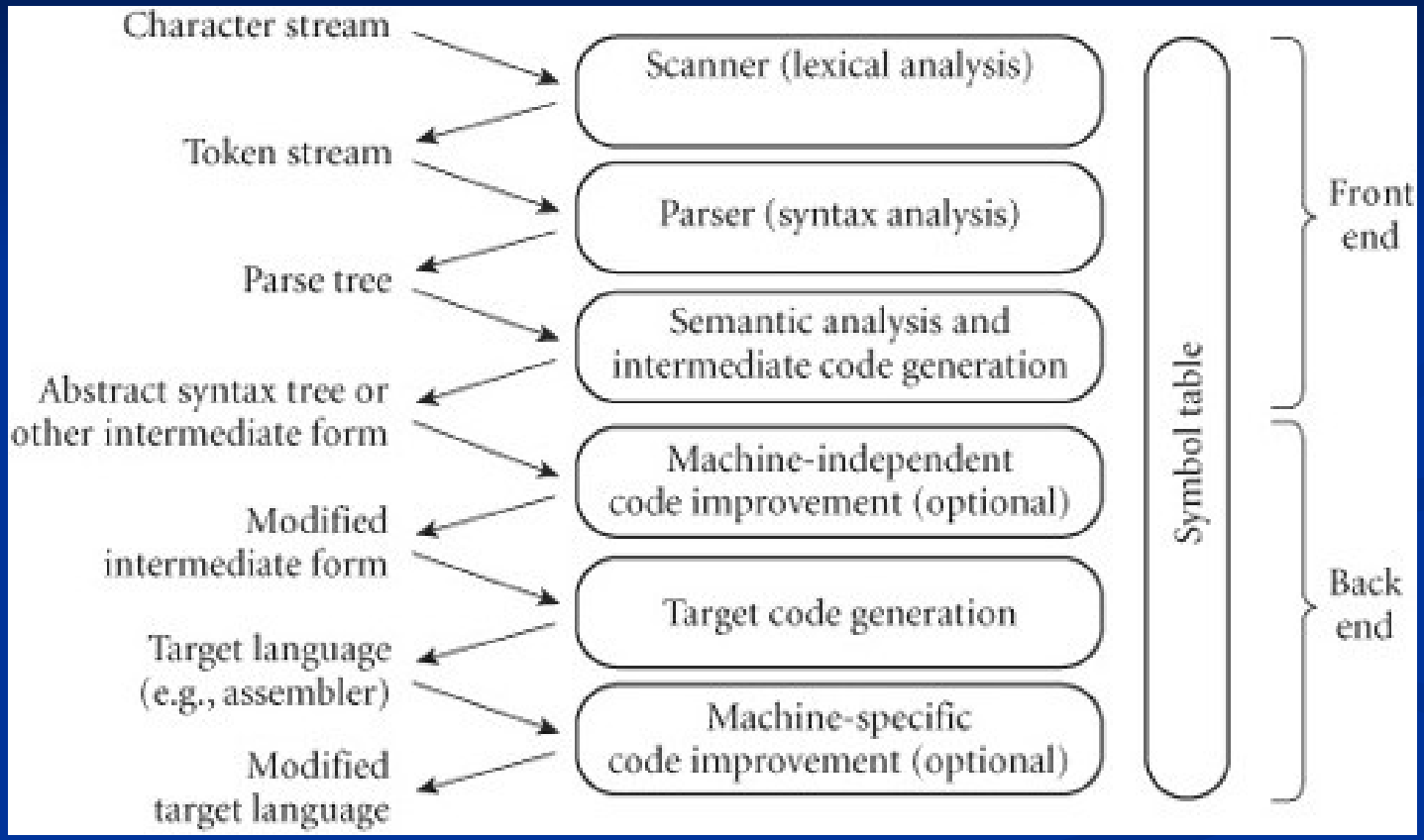
# Lots of other tricks

```
printf("The date is %s\n", __DATE__);
```

- Most preprocessor features are used for large/advanced software development practices

- gcc -E file.c > output.i

# Compiler?




* http://www.cs.montana.edu/~david.watson5/

- gcc -S

# Assembler

- Discussed in CS 250 – Computer Architecture
- gcc -c

  nm -v
- Really uses as

# Libraries

- Libraries are just collections of object files
  - Internal symbols are indexed for fast lookup by the linker
- Searched for symbols that aren't defined in the program
  - Symbol found, pull it into executable (static)
  - Otherwise include a pointer to the file, loaded by loader

# Statically linked

- Faster, to a degree
- Portable
- Larger binaries
- Fixed version, no updates
- File extension .a

# Dynamically linked

- More complexity
- Easy to upgrade libraries
  - Vulnerabilities
- Have to manage versions
- Loader re-links every time program is executed

```
readelf --dynamic /bin/ls
ldd /bin/ls
```

# End result

- gcc -o abs abs.c
  nm -v ./abs

# Loader

- Essential step in starting a program
- Historically allocated space for all sections of the executable (text, data, bss, etc)
- Now simply establishes mappings
  - Page faults actually populate the memory
  - For executable as well as (shared) libraries

- Also resolves any values in the executable to point to the functions/variables in the shared libraries
- Jumps to _start
  - init()'s all libraries
  - _then calls main()
  - ...and exit()
- Sometimes loaders are called "runtime linkers"

# Interpreter

readelf --headers /bin/ls

# Lazy binding

- Binding a function call to a library can be expensive
  - Have to go through code and replace the symbol with its address
- Delay until the call actually takes place
  - Calls stub PLT function
  - Invokes dynamic linker to load the function into memory and obtain real address
    - Rewrites address that the sub code references
    - Only happens once
- Procedure Lookup Table (PLT)

# Questions?