SEC204

Computer Architecture and Low Level Programming

Dr. Vasilios Kelefouras

Email: v.kelefouras@plymouth.ac.uk

Website:
https://www.plymouth.ac.uk/staff/vasilios-kelefouras

Date
30/10/2019

School of Computing
(University of Plymouth)

Overview

- **First, we'll briefly discuss development tools**
  - Assembler
  - Linker
  - Debugger
  - Compiler
  - Object code disassembler

- **Then we'll do practical activities**
  - *Please see Week_5b.pdf*

Assembler

- **ASSEMBLER**
  - Converts the Assembly language source code to machine/binary code for the processor
  - Assembly code varies for different assemblers (even for the same architecture)
  - *gcc file.c -S -o file.S //generate the assembly from C*
  - *gcc file.S -o file //generate the binary*
  - *gcc -c file.S -o file.o* //This will give object code file named file.o. It is binary but not executable
- Examples include
  - MASM
  - NASM
  - GAS
  - HLA

Linker

- The linker links together a number of object files to produce a binary file which can be directly executed
- Links objects: resolves all defined functions and memory address labels declared in the program code
- For external functions (e.g., printf) usually a second step is required to link the assembly object code with other external dynamic libraries and allow the executable to run on the host system
- *gcc file.o -o file* // Creates executable file from the object file test.o

Debugger

- Runs the program within its own controlled "sandbox"
- Runs the program in a controlled environment, specifying runtime parameters
- Stops the program at any point within the program
- Examines data elements, such as memory locations or registers
- Changes elements in the program while it is running to help bug removal
- We will be using GDB
- To tell GCC to emit extra information for use by a debugger, add -g to your other options.
  - *gcc test.c -o -g binary*

GDB Basic Commands

- break - Set a breakpoint
- watch - Set a watchpoint to stop execution when a variable reaches the specific value
- info - observe system elements, such as registers, the stack, memory
- x - examine memory location
- print - Display variable values
- Run - Start execution
- list - List specified functions or lines
- next - Step to the next instruction in the program
- step - Step to the next instruction in the program
- cont - Continue executing the program from the stopped point
- until - Run the program until it reaches the specified source code line (or greater)

COMPILER and DISASSEMBLER

- **COMPILER**
  - Converts high level code into assembly language and then into binary code for the processor to execute.
  - *gcc test.c -o test* //Creates executable file test from C language program test.c

- **DISASSEMBLER**
  - Takes a full executable program (or an object code file) and displays the assembly
  - *objdump -d binary_file* // shows the assembly of the binary file

Reverse Engineering (1)

- "Reverse engineering is the process of extracting the knowledge or design blueprints from anything man-made … is usually conducted to obtain missing knowledge, ideas, and design philosophy when such information is unavailable. In some cases, the information is owned by someone who isn't willing to share them. In other cases, the information has been lost or destroyed."
  – *Secrets of Reverse Engineering, Eldad Eilam*

Reverse Engineering (2)

- Reversing engineering your own software can show what others may learn from it
- If the software is supposed to "hide" information (e.g., encryption), make sure it works the way you think it should
- Preventing hackers from reverse engineering your code to find vulnerabilities
- Reverse engineering malicious software (malware) to prevent attacks
- Preventing stealing copyrighted information (digital rights)

Any questions?

Now let's open week5_b.pdf