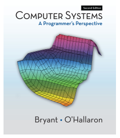

Computer Systems: A Programmer's Perspective, 2/E (CS:APP2e)

Randal E. Bryant and David R. O'Hallaron, Carnegie Mellon University

Lab Assignments

This page contains a complete set of turnkey labs for the CS:APP2e text. The labs all share some common features. Each lab is distributed in a self-contained tar file. You will need a CS:APP account to download the code. To untar `foo.tar`, type "`tar xvf foo.tar`" to the Unix shell. This will create a directory called "`foo`" that contains all of the material for the lab.

Handout directories for each lab (without solutions) are available to students who are using the book for self-study and who want to work on the labs. Solutions are provided only to instructors.

- *Data Lab* ***[Updated Sep 16, 2014]*** (README, Writeup, Release Notes, Self-Study Handout)

  Students implement simple logical, two's complement, and floating point functions, but using a highly restricted subset of C. For example, they might be asked to compute the absolute value of a number using only bit-level operations and straightline code. This lab helps students understand the bit-level representations of C data types and the bit-level behavior of the operations on data.

- *Bomb Lab* *[Updated April 9, 2013]* (README, Writeup, Release Notes, Self-Study Handout)

  A "binary bomb" is a program provided to students as an object code file. When run, it prompts the user to type in 6 different strings. If any of these is incorrect, the bomb ``explodes," printing an error message and logging the event on a grading server. Students must ``defuse" their own unique bomb by disassembling and reverse engineering the program to determine what the 6 strings should be. The lab teaches students to understand assembly language, and also forces them to learn how to use a debugger. It's also great fun. A legendary lab among the CMU undergrads.

  Here's a binary bomb that you can try out for yourself. The feature that notifies the grading server has been disabled, so feel free to explode this bomb with impunity. If you're an instructor with a CS:APP account, then you can download the solution.

- *Buffer Lab* ***[Updated Sep 10, 2014]*** (README, Writeup, Release Notes, Self-Study Handout)

  Students modify the run-time behavior of a binary executable by exploiting a buffer overflow bug. This lab teaches the students about the stack discipline and teaches them about the danger of writing code that is vulnerable to buffer overflow attacks.

- *Architecture Lab* *[Updated July 29, 2013]* (README, Writeup, Release Notes, Self-Study Handout)

  Students are given a small default Y86 array copying function and a working pipelined Y86 processor design that runs the copy function in some nominal number of clock cycles per array element (CPE). The students attempt to minimize the CPE by modifying both the function and the processor design. This gives the students a deep appreciation for the interactions between hardware and software.

  Note: The lab materials include the master source distribution of the Y86 processor simulators and the *Y86 Guide to Simulators*.

- *Cache Lab* ***[Updated Sep 2, 2014]*** (README, Writeup, Release Notes, Self-Study Handout)

  At CMU we use this lab in place of the Performance Lab. Students write a general-purpose cache simulator, and then optimize a small matrix transpose kernel to minimize the number of misses on a simulated cache. This lab uses the Valgrind tool to generate address traces.

  Note: This lab must be run on a 64-bit x86-64 system.

- *Performance Lab* ***[Updated Sep 2, 2014]*** (README, Writeup, Release Notes, Self-Study Handout)

  Students optimize the performance of an application kernel function such as convolution or matrix transposition. This lab provides a clear demonstration of the properties of cache memories and gives them experience with low-level program optimization.

- *Shell Lab* *[Updated July 28, 2003]* (Writeup, Release Notes, Self-Study Handout)

  Students implement their own simple Unix shell program with job control, including the `ctrl-c` and `ctrl-z` keystrokes, `fg`, `bg`, and `jobs` commands. This is the students' first introduction to application level concurrency, and gives them a clear idea of Unix process control, signals, and signal handling.

- *Malloc Lab* ***[Updated Sep 2, 2014]*** (Writeup, Release Notes, Self-Study Handout)

  Students implement their own versions of `malloc`, `free`, and `realloc`. This lab gives students a clear understanding of data layout and organization, and requires them to evaluate different trade-offs between space and time efficiency. One of our favorite labs. When students finish this one, they really understand pointers!

- *Proxy Lab* *[Updated Mar 26, 2003]* (Writeup, Release Notes, Self-Study Handout)

Students implement a concurrent Web proxy that sits between their browser and the rest of the World Wide Web. This lab exposes the students to the interesting world of network programming, and ties together many of the concepts from the course, such as byte ordering, process control, signals, signal handling, file I/O, concurrency, and synchronization.

Questions or problems with your account? Forget your password? Send mail to Dave O'Hallaron