Debuggers and Multi-Language Support Exercises CS 364 — Spring 2022

1 Definitions

1. Define the following terms, and give examples where appropriate.	
(a)	Application Kernel:
(b)	Breakpoint:
(2)	
()	
(c)	Conditional Breakpoint:
(d)	Debugger:
(e)	Debugging Table:
(f)	Marshalling:
(g)	Signal:
(3)	
(h)	Watchpoint:
(11)	waterpoint.

2 Debuggers

1. In class, we discussed the implementation of a debugger for a *compiled* language. How would this implementation change for an *interpreted* language? Describe the steps needed to stop the program on certain lines of code, step through expressions/statements, and access program state.

- 2. Write a program in C that reads numbers from the user in a loop. When the user presses ctrl-c, the program should print out the average of those numbers and exit. Hint: You can use our approach for catching segmentation faults (to implement breakpoints) to capture the SIGINT signal.
- 3. It can be nice to step backwards through code sometimes. For example, you might single-step "too far" and wish to go back one line. Unfortunately, it is difficult to "undo" instructions because of side-effects. Using your knowledge of debugger implementation, describe how you would implement a debugger that gives the illusion of "stepping backwards". (Hint: This feature is often called *replay debugging*.)

3 Multi-Language Projects

1. What are some of the reasons why a software project might contain multiple programming languages? At what point is a multi-language project advantageous? When is a multi-language project detrimental? 2. List and describe at least three (3) uses for the JNIEnv pointer provided as the first argument to JNI C functions. 3. Compare and contrast the JNI and Python's ctypes module. In particular, consider the examples from class and the relative length of both C implementations. Why is the C implementation for Python so much shorter than the implementation for Java? 4. Checksums are often used to validate data transmitted over a network. CRC32 is a common algorithm for computing a checksum. An implementation is provided here: https://web.mit.edu/freebsd/head/sys/libkern/crc32. c Download this file and replace line 49 (#include <sys/systm.h>) with #include <stdint.h>. Do not add to or modify this file in any other way. Write programs in both Python and Java that call calculate_crc32c (the last function in the file). Pass in 0 as the initial CRC32 value (the first argument).