CMPSC 440
Operating Systems
Spring 2014

Laboratory Assignment Six: Measuring the Size of Program Variables in C and Java

Introduction

Main memory, in the form of random access memory (RAM), is an important component of a modern computer. The memory management unit (MMU) of an operating system is responsible for allocating, deallocating, and compressing the memory required by the programs that the operating system is managing. One of the first steps towards understanding the management of memory is to learn more about the size of the primitive variables that programs ask the MMU to allocate to memory. In this laboratory assignment, you will implement both a C and a Java program to determine the size of the most prevalently used primitive types supported by these two languages.

Sizing Primitive Variables in C Language Programs

The C programming language provides a `sizeof` function that a program can use to determine the size of a primitive variable like an `int`. What are the inputs and outputs of the `sizeof` function? How does the `sizeof` function work? You should write a `usesizeof.c` program and run the `gcc` compiler to build a binary called `usesizeof`. Your program must calculate the size, in both bits and bytes, of the following variable types: `short`, `int`, `long`, `long long`, `float`, `double`, `long double`, `char`, `char*`, and `_Bool`. Do the sizes reported by your program make sense? Why?

Sizing Primitive Variables in Java Language Programs

While the Java language specification states that all primitive types should be uniform across implementations of the compiler and virtual machine, it is still worthwhile to determine the size of primitives in this language. However, the Java language does not provide a `sizeof` method like the one that you used in your C program. To learn more about how to calculate the size of a Java primitive, read the article called "Sizeof for Java: Object Sizing Revisited" by Vladimir Roubtsov. In the downloads section of this online article, you will find a Zip file called `02-qa-1226-sizeof.zip`.

After downloading this file, decompress it in the directory that you are using to store all of the files for this laboratory assignment. Using the example source code provide with this archive, you should write a Java program called `UseSizeOf.java` that calculates the size, in both bits and bytes, of the following variable types: `short`, `int`, `long`, `float`, `double`, `boolean`, and `char`. Don't forget that `UseSizeof.java` will not compile and the program will not run unless you have `objectprofiler.jar` in your `CLASSPATH` environment variable.

When you analyze the output of the `UseSizeOf` program, you should take into account that Java's autoboxing feature will transform your primitive types into objects. As such, your final output needs to consider the size of a managed object pointer. What are the sizes reported by your program? Do they make sense? Why? How do they compare to the sizes of the C variables?

Extending the Sizing Programs

So far, the programs that you wrote for this laboratory assignment focused on computing the size of primitive variables. You should add at least one advanced feature to both your C and Java sizing programs. For your C program, you could add code that can run `sizeof` on a `union`, `struct`, or another type of pointer. For your Java program, you could add code that sizes a complex Java object like a linked list. What do these results tell you about the size of non-primitive data types?

Analysis of the Sizing Results

Now that you have finished implementing an extended version of the sizing programs for the C and Java languages, you can reflect on the implications of their output. What do the results from running your programs suggest about the trade-offs in the management of memory pages in the entire memory address space? What are the differences associated with the management of primitive variables and pointers, dynamically allocated variables, or heap-resident objects?

To complete this laboratory assignment, you should write a report that explains the programs that you implemented and gives the full output from each program. Next, you should compare and contrast the results from the C and Java programs. Finally, you should make sure that you respond to all of the questions posed in the assignment. Please make sure that your report addresses how your results would influence the design of an operating system's memory management unit.

Summary of the Required Deliverables

This assignment invites you to submit printed and signed versions of the following deliverables:

1. The properly commented source code of the `usesizeof.c` and `UseSizeOf.java` programs
2. The command lines that you used to compile and run the C and Java programs
3. A comprehensive report that explains the results and responds to all of the stated questions
4. Suggestions for additional features to add to the variable sizing programs
5. Ideas for future experiments and analyses that use the variable sizing programs
6. A reflective discussion of the challenges that you encountered when completing this assignment

In adherence to the honor code, students should complete this assignment on an individual basis. While it is appropriate for students in this class to have high-level conversations about the assignment, it is necessary to distinguish carefully between the student who discusses the principles underlying a problem with others and the student who produces assignments that are identical to, or merely variations on, someone else's work. As such, deliverables that are nearly identical to the work of others will be taken as evidence of violating the Honor Code.