

## Memory Debugging with Pin

Rather than have you implement memory debugging using recompilation and a set of C preprocessor macros, this assignment will introduce you to a fairly useful dynamic binary instrumentation tool developed by Intel. The Pin tool allows you to statically and/or dynamically modify a program to add and remove instrumentation in the form of general callbacks to your code. In an extreme case, you could have your code executed before each instruction in the program, but obviously this kind of interaction produces a dramatic slowdown. Pin is written in C++, so you'll also get a fairly gentle introduction to C++ interfaces in case you haven't used the language before.

The tool and documentation is available at

[www.pintool.org](http://www.pintool.org)

The assignment as a whole will consist of a week of design, a week or two of implementation, and a week of application. This document outlines the design, which you'll turn in during class next Tuesday. I'll take a look and highlight what I see as the best ideas on Thursday, after which everyone can implement the best strategy (*i.e.*, you're free to use other people's ideas from the design).

You probably won't actually need to download Pin itself until you start implementing, but feel free to grab a copy and get started or simply play around with it if you think it'll help. If you do write start writing code, be aware that I will limit groups to **three people**, so don't pass code around unless you know with whom you're going to work.

Your task in this homework is to brainstorm about the kinds of errors that people make with dynamic memory management and how you can detect and identify them using a tool like Pin.

### Memory Management Errors

To get you started, think about the lifetime of a block of memory in a fairly simple memory allocation library that does not join or split blocks. That is, once a block is allocated, assume that the block retains its size until the program terminates.

When a program starts, most of the virtual address space is not mapped, so we'll assume that the block starts out in an unmapped region, and that the first event in its lifetime is an initial mapping into the program's address space, at which point the block is owned by the dynamic allocation library. Events after the mapping consist of a sequence of alternating mallocs and frees. At least, such is the order when a block is used properly.

Your first task is to identify all possible types of errors made by programs using dynamic allocation. Organize these errors in some meaningful way and describe each type. Be sure to think carefully here; in the next section, you'll consider each error in turn, and missing one here means that you won't mention it later.

## Debug Strategy

For each kind of error that can be made (remember, it's up to you to create a meaningful error taxonomy), answer the following questions:

- What action constitutes the error (instruction, function call, *etc.*)?
- Using a tool such as Pin, can you *guarantee* that this kind of error is detected? If so, explain how, being explicit about the types of Pin capabilities that you'll use. If not, give an example that cannot be detected.
- Can you guarantee detection while incurring an overhead that might be considered tolerable in a production environment (let's say 5-10% slowdown)? If not, or if no guaranteed approach exists, outline an alternative with reasonable overhead and estimate the probability that it detects a given error (a property known as error coverage).
- When you detect this kind of error using your practical (and possibly guaranteed) scheme, what kind of information can you provide to help the programmer identify the source of the problem (*i.e.*, the bug that leads to the error)?