Premalloc:

Pre-Runtime Heap Segmentation for Dynamic Memory Allocation

Jake Masters, Dr. Jonathan Geisler

Taylor University

Introduction

Dynamic memory allocation involves storing and managing chunks of memory at the request of a running computer program. A program can request a memory chunk of any size at any point in time, which implies that management is non-deterministic. One of the most widely used memory allocator implementations, `dlmalloc` (based on Doug Lea's algorithm), satisfies requests for dynamic memory chunks by algorithmically populating data structures known as tree bins, which each store various numbers of same-sized chunks [1]. However, during a computer systems classroom discussion, we hypothesized that given a program's source code or runtime data, instruction overhead can be reduced. This research presents a new dynamic memory allocation algorithm we call `premalloc`.

Background

We have limited the current scope of our research to dynamic memory allocation for Linux processes, which employ a section of the virtual address space known as the **heap**.

With respect to the algorithm, all chunks in both `dlmalloc` and `premalloc` are stored in a number of bins as illustrated below. Both implementations desire to populate all bins in such a way that they can satisfy various memory requests. In Figure 1, this end state is visualized on the right side of both arrows.

The difference between the two algorithms is `dlmalloc` begins with one chunk the size of the entire heap, while `premalloc` prepopulates each bin with what it believes is a *best-fit population* based on the information it is given. Each algorithm's initial state is given on the left side of Figure 1.

Current State

Currently we are engaging in three threads of work:

- Defining and understanding relevant metrics
- Understanding memory allocation mechanisms
- Implementing our own memory allocator

We have completed preliminary best and worst-case analysis on the performance of `dlmalloc` using the Linux `perf` tool. We have studied the policies and mechanisms behind `dlmalloc`, which is composed of over **six thousand lines** of source code [2]. Finally, We are currently building `premalloc`, which is the software that will be our primary deliverable for this semester of research.

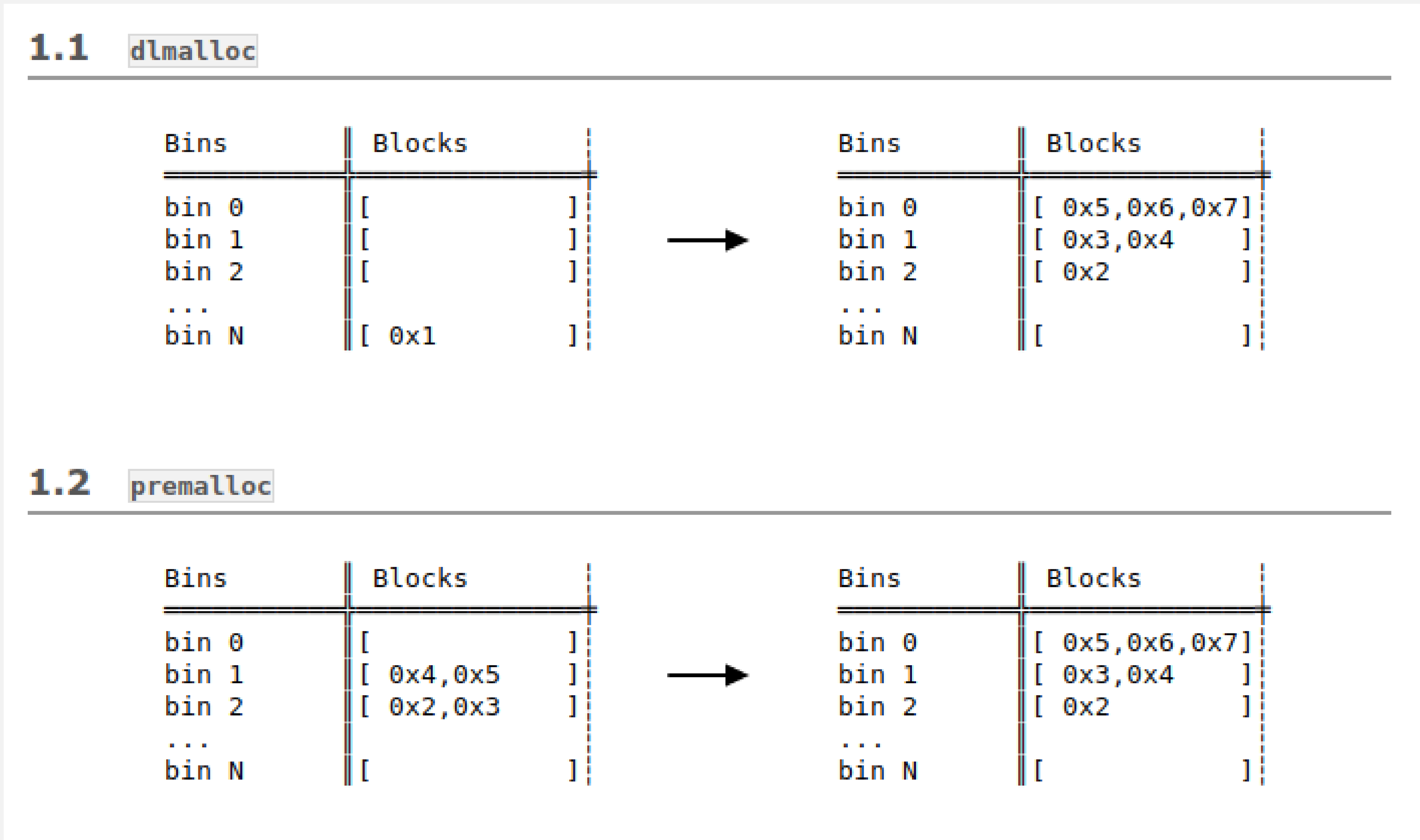

Figure 1: Both `dlmalloc` and `premalloc` need to arrive at the same end state to satisfy memory requests. The initial state for `premalloc` requires fewer instructions.

Future Work

After our implementation of `premalloc` is in a working state, we will be conducting experiments in order to test the effectiveness of several methods in producing pre-populated free lists. Our principal metric of success is the **CPU time spent** executing memory allocation code using pre-populated free lists as opposed to algorithmically populating them at runtime. We chose this metric because it accurately measures the reduced overhead incurred by `premalloc` compared to `dlmalloc` [3]. Figure 2 illustrates the proposed process of gathering information about dynamic memory usage, in this case at runtime, to inform the memory allocator in order to pre-populate for subsequent executions.

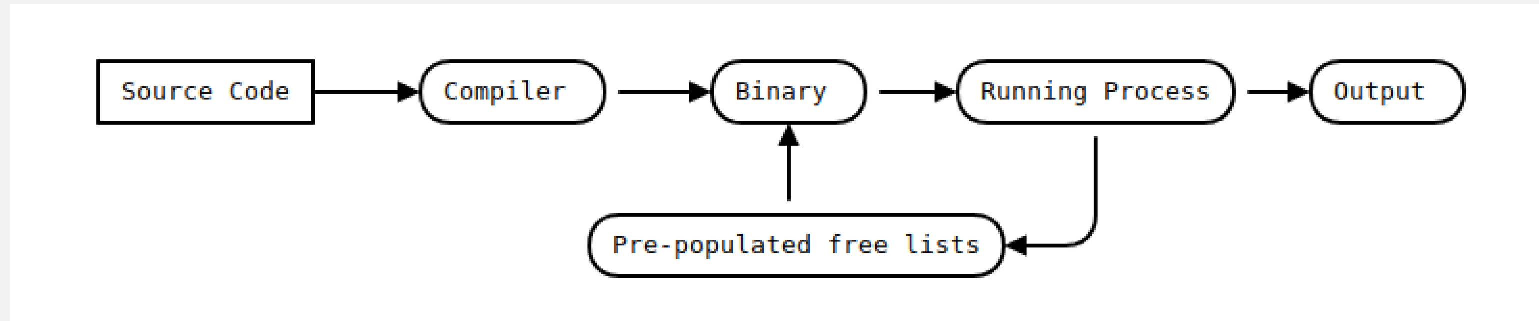

Figure 2: On the first run, runtime data is generated for a given program, which is fed back into premalloc on subsequent runs.

References

[1] P. Wilson and et al.
Dynamic Storage Allocation: A Survey and Critical Review.

[2] W. Fang.
Analysis on Dynamic Memory Allocation.

[3] E. Berger and et al.
Reconsidering Dynamic Memory Allocation.

Acknowledgements

We would like to thank Dr. Denning for his service and role in supervising the computer science research program in our department.