Csc344 Problem Set: Memory Management / Perspectives on Rust ((Template))

Task 1 - The Runtime Stack and the Heap

Here, we will be discussing how memory flows on a computer in order to gain a deeper knowledge of memory management. Stack and Heap are very important components of a Computer system. As a programmer, we have to have a deep understanding of how memory management functions. In the following two paragraphs, we will try to explain both the concepts, Stack and Heap.

The runtime stack is a concept used by computer programs, where the program allocates memory on the stack to store the function's arguments and local variables. The stack operates on a "Last In, First Out"(LIFO) principle, which means that the last function called is the first one to one to return. It is an essential part of most programming languages and understanding how it works is crucial for debugging and optimizing code.

Unlike Stack, heap is a fixed-sized data structure and is not automatically managed by the program and is managed by the runtime environment. When a program requests memory from the heap, the operating system allocates a block of memory from the heap and returns a pointer to the beginning of that block. The program can then use this pointer to access and manipulate the memory in the block. The program is responsible for managing the heap, including allocating and deallocating memory as needed. If memory is allocated from the heap but not released when it is no longer needed, the program may suffer from memory leaks, which can cause the program to consume more and more memory until it crashes or runs out of memory.

Task 2 - Explicit Memory Allocation/Deallocation vs Garbage Collection

There are two major approaches to memory management crucial for a programmer: Explicit Memory Allocation/Deallocation and Garbage Collection. We will discuss these two concepts in detail in the next two paragraphs. Both these concepts are crucial depending on the language you're using but the more we know, the better programmers we become,

Explicit Memory Allocation/Deallocation is a way to manually manage the memory and it requires the programmer to explicitly allocate and deallocate memory using programming constructs provided by the language. Once the program no longer needs the allocated memory, the programmer must explicitly deallocate it. Failing to deallocate memory can result in memory leaks as well. This method gives the programmer control over memory management, which can be very important in performance-critical applications but it also requires the programmer to be responsible for managing memory correctly.

Garbage collection, on the other hand, is an automatic management technique used in programming to manage memory on the heap. Unlike explicit memory allocation and deallocation, garbage collection automatically identifies and frees the memory that is no longer needed by the program. It is widely used in programming languages like Java, Python, C#, and others. Garbage collection does prevent memory leaks and dangling pointers but it has its disadvantages. The garbage collector must periodically scan the heap, which can consume significant CPU resources and impact performance. Also, it could introduce unpredictable pauses in the program execution, which might hinder consistent performance.

Task 3 - Rust: Memory Management

1. Rust allows high control in memory usage due to the allocation and deallocation of memory through the heap.
2. Without the need for a garbage collector, Rust ensures memory safety using the concept of ownership and borrowing.
3. Once the block ends, the variable is out of scope. In Rust, when we declare the variable within a block, we can not access it after the block ends.
4. In Rust, we define memory cleanup for an object by declaring the drop function unlike in C++ where we had to call the delete.
5. Primitive types of Rust can be copied since they have a fixed size and they live on the stack.
6. Memory can only have one owner in Rust.
7. Passing variables to a function gives up ownership. For example, if you pass a variable to a certain function, you can no longer use it.
8. Since Rust is very performance-oriented, it avoids using deep copying by default. If a and b go out of scope, both variables will be dropped.
9. We can pass variables by reference. It allows another function to “borrow” ownership rather than taking it so that the original still has access to it after the function is done using it.
10. Slices are like an array or a vector and must be primitive data, stored on the stack or should be a reference to another object. Slices will never have ownership of the heap and hence, Rust will not deallocate the memory even when they’re out of scope.

Task 4 - Paper Review: Secure PL Adoption and Rust

Rust was developed by Mozilla to be a practical and secure alternative to C and C++, which is low-level, and type- and memory safe. Rust's "zero-cost abstractions" and its lack of garbage collection make it appropriate for resource-constrained environments. Although C and C++ continue to be widely used, Rust has been getting popular recently. The new users of Rust and the veterans seem to have very similar experiences, which suggests consistency.

Rust has been perceived to succeed at its goals of security and performance. There is also a very active community and they agreed on high-quality documentation and clear error messages to be a key strength of Rust. As a low-level programming language, Rust has been known to extend the programmer's knowledge of models of secure programming. Rust also supports generics and modules, and a sophisticated macro system, and its variables are immutable by default. Most people learn Rust because it's either interesting or in demand. There are not a lot of Rust programmers because it is also extremely hard. Rust does boost your confidence in other languages because you understand the basics of programming since Rust does not have a lot of libraries, you are forced to learn the basics on your own.

Talking about drawbacks, Rust does have easy-to-use tools, but they are very slow. Rust also has a very steep learning curve, especially for beginners. The borrow checker and programming paradigms are the hardest to learn. The lack of libraries and infrastructures causes dependency bloat. Although there are some high-quality tools available, Rust lacks some critical libraries. It is also really hard to hire a Rust developer, which discourages a team to use Rust as a major platform. These issues are fixable and that is only possible if Rust gets enough recognition in this programming world. Rust has a lot to offer and its difficulty level will definitely help programmers

improve their knowledge on memory handling and programming in general.