Assignment One – Erlang Questions

Jason Gasparini
Jason.Gasparini@Marist.edu

September 9, 2023

1 Erlang and its Basics

1.1 What is Single Assignment?

Single assignment is the description of the behavior relating to how one sets variables and their values in Erlang. Dealing with single assignment variables means that one cannot reassign a variable to a different value. This is unique when compared to some of the most popular programming languages used today. In order to "reassign" a variable, it must be unbound from its original value.

1.2 Bound vs. Unbound variables

The difference between Bound and Unbound variables is the way in which they can be used. Variables in erlang are essentially expressions, the returned value of a bound variable will always be the value that it has been "assigned" or bound to. Unbound variables are used for pattern matching cases as they are "free" of a value and can be set within pattern matching expressions.

1.3 Variable Scope

The scope for variables is naturally associated with the function clause in which it is defined/used. If a variable is used in a branch or if case, it must be defined in all branches in order to retain its value.

1.4 Does erlang use mutable or immutable memory state?

Erlang implements immutable memory state and all data structures are immutable. Erlang's processes are isolated and the immutability means that one process will not be able to affect the data from another process.

1.5 Erlang's memory management

Processes in erlang have individuality in regards to their heaps. Each erlang process has its own private heap used to store data. Erlang also uses a "generational" garbage collection strategy. Frequently changing data and objects are allocated in the "young generation" while longer lasting objects will get moved to the "old generation" heaps. This helps with efficient memory management.

1.6 "Erlang" ???

The naming for Erlang is a reference to Agner Krarup Erlang who was a Danish mathematician and engineer who made important contributions to the study of telephone traffic. This was fitting when considering the telecommunication applications and goals of the language.

1.7 Soft real time vs. Hard real time

In programming languages, soft real-time and hard real-time describe approaches to handling time-critical tasks. Soft real-time systems are those in which meeting deadlines are "important but not critical". This means that occasional missed deadlines are acceptable if the system continues to function well. Multimedia streaming is an example of a "soft real-time" system and makes sense when one considers that the data transmission of this system uses UDP. Hard real-time systems are the opposite in the way that they have strict and non-negotiable timing requirements and deadlines. This means that any failure to meet a deadline for a task can have detrimental consequences such as system failure.

1.8 Concurrency-oriented programming

Erlang is well suited for concurrency-oriented programming due to its characteristics such as its lightweight processes and garbage collection strategy. Its lightweight processes are able to be created and managed in larger numbers compared to to "heavyweight" threads required by other languages. Its isolated memory also allows for concurrency without the many concurrency bugs that other languages would create. As described before, its generational garbage collection strategy allows for concurrency efficiency.

1.9 Let it crash!

"Let it crash" is Erlang's approach to building "fault-tolerant" systems. It's defined by the way that Erlang handles errors and failures. The idea behind this philosophy is that it is more practical to handily and "gracefully" manage errors rather than trying to anticipate and create cases for every failure that could occur. This is relatable to the soft real-time applications of the language.

1.10 Tuple vs. List

While tuples and lists are both immutable data structures in Erlang, tuples are typically used when one is dealing with a fixed group of elements. Tuples can contain elements of different types while lists can only hold the same type. Even though both are immutable if one wanted to add an element to their group, they would add to a list by recreating it and appending the new element. Tuples are generally faster at accessing their elements because of their fixed structure and are often used in pattern matching expressions.

1.11 BEAM

Bogdan's Erlang Abstract Machine (BEAM) is the runtime environment utilized by Erlang. BEAM is to Erlang as the JVM is to Java. BEAM is what is responsible for all of the characteristics of Erlang and is essentially what brings the language to life. It is what allows for the process management of Erlang and its concurrency. The BEAM virtual machine is what is responsible for executing Erlang code.

1.12 More Erlang process than allowed for

Creating more processes than what the OS allows for can be attributed to the power of the BEAM VM. Its Erlang scheduler, which is what manages the execution of Erlang's processes, is able to multiplex a lot of Erlang processes onto smaller amounts of OS-level threads and CPU cores. Each thread can handle thousands of processes and this allows for Erlang to create concurrent processes without having to one-to-one map to OS threads.