Roshan Poudel

1.

- What does the flush function do? Why might we want to use it? What module is it in?

> Flush function is used to print and clear all messages from the mailbox of the current process. We can use flush for debugging purposes and also clearing the mailbox. Also, when working with processes in iex, flush() helps to quickly view messages in the mailbox without writing a receive block. flush/0 is not in a standard Elixir module like Kernel. Instead, it is part of iex (Interactive Elixir Shell).

- How can a process send a message to itself (right in IEX or otherwise)?

> Process can send a message to itself by referring to its own PID using self(). eg: send(self(), message). In regular processes, it enables stateful recursion and periodic behavior.

- What is the purpose of having linked processes?

> The purpose of linked processes is to create a relationship between processes so that they can monitor and react to each other's failures. For eg, if we want the parent process to terminate if one of the intermediate process fail, we should use the linked processes. They enable error propagation, fault isolation, and robust process management.

3.

- What two key strengths of the language are identified?

> Concurrency and fault-tolerance

- According to the interviewees, what are the types of cases where Erlang "just works very well"

> Server Development and Telecommunications

- Robert Virding says they didn't set out to build a functional language. Why then was making Erlang a functional programming language the right approach?

> Adopting a functional approach, which emphasizes eliminating side effects, naturally facilitated concurrency and fault tolerance. This design choice also simplified garbage collection by avoiding complex backward references. That's why functional programming was a right approach.

- The guys mention Amdahl's Law (which Virding calls "depressing"). I think Dave Thomas mentioned it in the video we saw first day of class. Look it up and explain what the Law tells us about our programs.

> Amdahl's Law, mentioned by Virding as "depressing," addresses the limitations of parallel processing. It states that the maximum speedup of a program is constrained by the portion that cannot be parallelized. For example, if 5% of a program is inherently sequential, the theoretical maximum speedup, even with infinite processors, is 20 times. Amdahl's law can be formulated in the following way:

$$S_{\text{latency}}(s) = \frac{1}{(1-p) + \frac{p}{s}}$$

> where:

> S_{latency} is the theoretical speedup of the execution of the whole task; s is the speedup of the part of the task that benefits from improved system resources; p is the proportion of execution time that the part benefiting from improved resources originally occupied.

- Starting at 30:11 they start to talk about cloud computing, research into massively multicore systems, and the notion of scale change in computing, including speculation about future technologies enabled by scale changes to come. Describe one past and one future "scale change" and what tech these changes enabled or perhaps will enable.

> The transition from single-core to multicore processors allowed for parallel execution of tasks, significantly enhancing performance and enabling complex applications like real-time data processing. The anticipated development of processors with thousands of cores could revolutionize computing by facilitating massive parallelism, potentially leading to advancements in artificial intelligence and real-time simulations

4.

- Are there any types of data that it fails to receive and output?

> No. All the listed types (int, tuple, atom, list, map, and PID) are supported in Elixir's messaging system.

- All other required changes are made in the file concurrent.exs

5.

- Describe what is returned (not just the type) by each function called in the pipeline.

> 1..num processes creates a range from 1 to num_processes. Then it is passed to Enum.map. For each element in the range, it spawns a new process using the spawn/3 function, which calls

> the func from the module with the current process's PID [self()] as an argument and that returns a list of PIDs of the spawned processes, which is then passed to schedule_processes along with the to_calculate list and an empty list for results. The schedule_processes function processes these inputs and returns the final sorted list of results.

- Timing Results

```
 1 4.17
 2 1.79
 3 1.31
 4 0.94
 5 1.01
 6 0.97
 7 0.84
 8 1.00
 9 0.76
10 0.80
```

- SpeedUp Calculations:

> SpeedUp = $\frac{time\ with\ 1\ process}{time\ with\ n\ processes}$

> For 2 processes: SpeedUp = $\frac{4.17}{1.79} = 2.33$

> For 4 processes: SpeedUp = $\frac{4.17}{0.94} = 4.44$

> For 10 processes: SpeedUp = $\frac{4.17}{0.80} = 5.2$

- What operating system, what processor, and how many cores?

> MacOS, Apple M2 processor, 8 cores

- What is the highest (approximate) scheduler utilization you see?

> Near 100 % Scheduler Utilization