**Developing a Simple Kernel Module Interfacing with Java**

Thomas Kojo Quarshie & Richard Kweenu Quayson

Ashesi University

Dr. Tatenda Kavu

CS433\_B: Final Project Submission

April 16, 2024

**JavaFX Application**

The JavaFX application was developed to give users the opportunity to interact with the kernel in managing files and folders in the user space. The JavaFX application followed a rigorous MVC architecture pattern – Model, View, and Controllers, separating the concerns of data management, user interface, and application logic.

For the user space (Frontend) aspect of the application, we created FXML files for defining the layout and components of the user interface. We also utilized common JavaFX UI components such as TableView, TableColumn, Label, Button, etc., to interact with users. We implemented functionalities to create, edit, and delete files and folders, and displayed file details such as name, type, path, creation date, modification date, and size for the user to see.  
  
**Lessons Learnt from JavaFX Application:**

1. We gained familiarity with the JavaFX framework and its capabilities for building interactive desktop applications with rich user interfaces.
2. We learned about the MVC design pattern and how to structure JavaFX applications for better organization, maintainability, and scalability.
3. We learned how to interact with the file system using Java, including creating, renaming, and deleting files and folders programmatically.
4. We also gained experience in integrating Java applications with external systems, such as kernel modules, to leverage additional functionalities.

**Challenges faced:**

We faced a lot of challenges, especially with the JavaFX application, since we were new to it. It was difficult integrating the right packages (i.e., JAVA SDK and user defined libraries containing external JAR files) into the application, which required a lot of research. Moreover, with the new eclipse released, it is very difficult to integrate it with FXML files for easier development of the interface with Scenebuilder. As such, we read a lot of articles to probe more into this issue and how to solve it. We finally got it resolved by downloading an xml file online and adding it as a package to our JavaFX application.

Aside from the challenges with the JavaFX application, we also found it difficult integrating our kernel module developed using C code into our JavaFX application. We read online and tried so many times, but we were facing issues in obtaining both the file and directory data such as filename, size, and type. This is because reading and writing from files within the Linux kernel has been wrapped within the virtual file system it provides and delegated for user space applications and not the kernel itself. As a result, we faced issues trying to utilise the vfs\_stat and vfs\_fstatat functions to read information about a file since such functions are no longer exported for use in kernel modules.

**KERNEL MODULE DEVELOPMENT (C CODE):**

This is the aspect where most of our efforts and development went into. We developed a kernel module that gives users permission to create, rename, delete, and retrieve the details of both files and folders respectively.

**Essential linux commands learnt:**

sudo insmod <kernel\_module\_name> -> insert kernel module process

sudo rmmod <kernel\_module\_name> -> removing kernel module process

sudo dmesg -> checking kernel logs

sudo dmesg | tail -> retrieve last messages from kernel logs

lsmod | head -> retrieve recently added kernel modules

lsmod | grep <kernel\_module\_name> -> check for a particular kernel module's process

**Overall lessons learnt:**

The project served as a profound reminder of the indispensable value of kernel logging in the software development lifecycle. As we delved into creating and debugging custom kernel modules, we quickly realized that kernel logging was not merely an auxiliary tool but the backbone of our debugging efforts. Despite the initial reluctance towards delving into error logs, we soon recognized them as an invaluable asset, offering a treasure trove of insights into system behavior and intricacies. These logs became our guiding light, providing detailed accounts of system events, error messages, and warnings, which were instrumental in diagnosing and resolving issues effectively.

In essence, our journey with kernel logging underscored its significance as a fundamental aspect of software engineering. It taught us the importance of embracing error logs not as burdensome chores but as essential companions in the pursuit of robust software solutions. By acknowledging error logs as indispensable companions in our development process, we cultivated a mindset of proactive problem-solving and continuous improvement. This shift in perspective empowered us to harness the full potential of kernel logging, transforming what might seem like mundane logs into powerful tools for enhancing system reliability, diagnosing issues, and fostering a deeper understanding of system behavior.