SNHU \_CS\_405\_Secure coding

**C/C++ Emerging System Architectures & Technologies Portfolio**

**Overview**

This portfolio shows various projects and implementations using C/C++ that explore and demonstrate modern system architectures and emerging technologies. The focus is on efficient, scalable, and maintainable solutions leveraging contemporary design principles and best practices.

**Key Highlights**

**1. Project: Multi-Threaded File Processor**

* **Description**: Implements a high-performance file processing tool using multithreading to optimize I/O operations.
* **Features**:
  + Thread pool implementation for parallel processing.
  + Optimized for large datasets.
* **Best Practices Used**:
  + Mutexes and condition variables to ensure thread safety.
  + Leveraged RAII for resource management.
* **Technologies**: POSIX threads, STL.

**2. Project: Embedded System Simulator**

* **Description**: Simulates a basic embedded system environment for sensor data processing.
* **Features**:
  + Real-time data collection and analysis.
  + Emulated UART communication.
* **Best Practices Used**:
  + Memory-efficient design using smart pointers.
  + Use of hardware abstraction layers for portability.
* **Technologies**: Standard C++, Embedded C concepts.

**3. Project: Custom Memory Allocator**

* **Description**: A custom memory management system designed for low-latency applications.
* **Features**:
  + Fixed-size block allocation.
  + Garbage collection simulation.
* **Best Practices Used**:
  + Minimization of heap fragmentation.
  + Extensive unit testing.
* **Technologies**: Low-level C, Valgrind for memory leak checks.

**4. Project: Distributed Key-Value Store**

* **Description**: A scalable distributed key-value store.
* **Features**:
  + Consistent hashing for data distribution.
  + Leader election using RAFT algorithm.
* **Best Practices Used**:
  + Modular architecture for scalability.
  + Unit and integration testing.
* **Technologies**: C++, Boost library,

**Best Practices in C/C++ Development**

**1. Code Quality**

* Followed **Google C++ Style Guide** for consistent code formatting.
* Used **Clang-Tidy** and **CppCheck** for static code analysis.

**2. Performance Optimization**

* Profiling with **gprof** and **Valgrind**.
* Memory management using smart pointers and custom allocators.

**3. Testing and Debugging**

* Unit testing with **Google Test Framework**.
* Debugging with **gdb** and logging with **spdlog**.

**4. Modern C++ Features**

* Utilized features like lambda expressions, range-based loops, and type inference for cleaner and more expressive code.
* Applied **std::thread** and **std::async** for concurrency.

**5. Documentation and Version Control**

* Well-documented code using Doxygen.
* Version control managed with Git and GitHub.

**How to Use**

**Prerequisites**

* C++17 or later.
* CMake 3.16+.
* Compatible compilers: GCC, Clang, or MSVC.