****

**UNIVERSITY OF GHANA**

**SCHOOL OF ENGINEERING SCIENCES**

**COLLEGE OF BASIC AND APPLIED SCIENCES**

**DEPARTMENT OF COMPUTER ENGINEERING**

**FIRST SEMESTER 2023/2024 ACADEMIC YEAR**

**COURSE CODE: CPEN 307 - OPERATING SYSTEMS**

**COURSE INSTRUCTOR: DR.GIFTY OSEI**

**LAB: 2**

**NAME:**

DOE AGUDEY DANIEL

**ID:**

10956661

**DATE**: November 4, 2023

**SYSTEM CALL INTERFACE IMPLEMENTATION**

**ABSTRACT**

This lab report outlines the design and implementation of a system call interface for a kernel space program. The system call interface is created to facilitate user interactions, input validation, and memory reporting. The report includes a detailed description of the objectives, methodology, and outcomes of the lab exercise, including a testing phase with accompanying screenshots.

**INTRODUCTION**

This lab focused on the integration of a system call interface that connects the user interface with the kernel space of an operating system. Independently developed, this interface is tailored to manage user interactions, validate input, and report memory locations. The exercise emphasized the importance of code modularity, header file usage, regular expressions for input validation, and the establishment of seamless communication between the user interface and the kernel space, reflecting key aspects of operating system design.

**METHODOLOGY**

**1. Refactoring the Kernel Space Program**

The existing kernel space program was refactored to improve code modularity and organization. The following functions were introduced:

* **generateRandomNumbers():** This function generates random numbers to populate cache1 and cache2.
* **searchMemory(int target):** A function to search for a specific integer in cache1, cache2, and RAM.
* **replaceInCache(int target):** This function replaces the first element of cache1 with the searched integer if it is not found in cache1.

**2. Creating the System Call Interface**

**2a. Accepting User Input**

A function was defined in 'systemCall.h' to accept user input:

int AcceptUserInput();

**2b. Input Validation with Regular Expressions**

A function was defined to check the validity of user input using regular expressions to ensure it is not a string or special character:

bool IsValidInput(int input);

**2c. Reporting Memory Locations**

Another function was defined to inform the user about the memory location of the searched number and display the contents of cache1, cache2, and RAM:

if (IsValidInput(userInput)) {

bool foundInCache1 = SearchInCache(userInput, cache1, CACHE1\_SIZE);

bool foundInCache2 = SearchInCache(userInput, cache2, CACHE2\_SIZE);

bool foundInRam = SearchInRam(userInput);  
……}

**3. Implementing the System Call Interface**

The 'systemCall.cpp' file contains implementations for the functions defined in 'systemCall.h'. These implementations include handling user input, validation, and reporting memory locations.

**4. Integrating the System Call Interface**

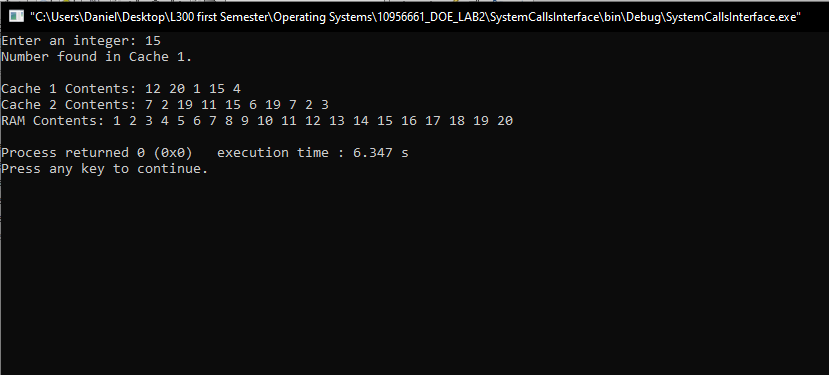
The kernel space program integrates with the system call interface by including 'systemCall.h'. The provided functions are used for user input, validation, and memory reporting.

**RESULTS**

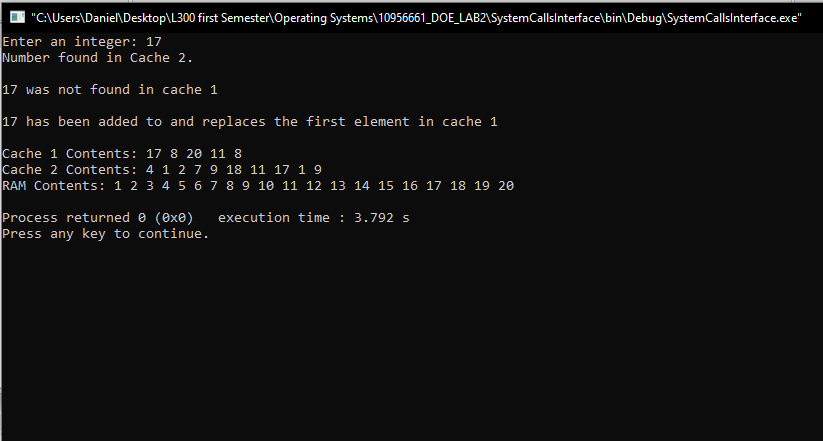
The system call interface implementation was successful and enabled user-friendly interactions with the kernel space program. Users could input data, and the system verified the input's validity, ensuring it was not a string or a special character. The system also accurately reported the memory location of the searched number and displayed the contents of cache1, cache2, and RAM.

**Testing Phase**

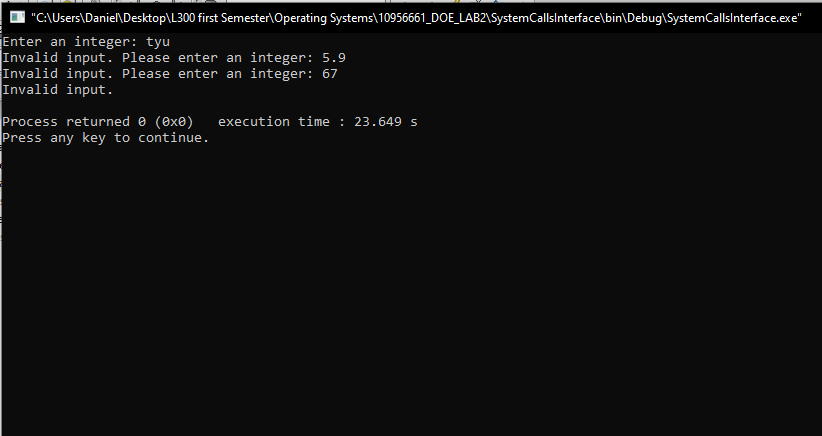
The system underwent a testing phase to ensure its functionality. Below are screenshots demonstrating the system in action:



*Figure 1: Interger found in cache*



*Figure 2: when integer is not found in cache 1*



*Figure 3: When a wrong value is entered*

**CONCLUSION**

The system call interface's successful implementation vividly mimics the pivotal interaction between user interface and kernel space. This achievement underscores the essential principles of code modularity, header files, and seamless communication between system components. The system is primed for practical application, facilitating an authentic and user-friendly experience that parallels real-world operating systems. This hands-on experience has provided a deep understanding of the crucial interplay between user interfaces and the kernel space in operating systems.