###### 

***File Navigation System***

**Submitted By:**

**Salahuddin Nasir**

**Roll Number:**

**G1F22UBSCS031**

**Course Instructor:**

**Prof. Hassan Rauf**

## File Navigation System:

### 1. Memory Management Enhancement:

We've expanded the memory simulation with the ability to allocate and deallocate memory blocks, as well as display memory usage.

* ****MemoryBlock Structure:****

c

typedef struct {

int size;

int is\_free;

int start\_address;

} MemoryBlock;

* ****Global Variables:****

c

MemoryBlock memory[MEMORY\_SIZE];

int memory\_blocks = 0;

* ****Functions:****
  + allocate\_memory(int size)**:** Allocates a memory block of the specified size if available.
  + deallocate\_memory(int block\_number)**:** Frees the specified memory block.
  + display\_memory()**:** Displays the current memory blocks, their sizes, and statuses.

### 2. Advanced Process Scheduling:

The process scheduler now includes process states and simulates different scheduling algorithms.

* ****Process States:****

c

typedef enum {

NEW,

READY,

RUNNING,

WAITING,

TERMINATED

} ProcessState;

* ****Process Structure Update:****

c

typedef struct {

int pid;

char name[256];

ProcessState state;

int burst\_time;

} Process;

* execute\_process(void \*arg)**:** Simulates a process progressing through states from RUNNING to TERMINATED based on its burst time.
* schedule\_processes(Process processes[], int count)**:** Uses Round Robin scheduling to execute processes, considering their states and burst times.

### 3. Security Improvements:

* ****User Authentication:****
  + authenticate\_user(int \*authenticated)**:** Prompts the user for a username and password. For demonstration purposes, the correct password is hard-coded as "password123". In a real-world scenario, you'd verify credentials securely.
* ****Password Input Without Echo:****
  + disable\_echo() **and** enable\_echo()**:** Functions to disable and enable terminal echo so that the password input isn't displayed on the screen.

### 4. Error Handling and Input Validation:

* ****Error Handling Function:****

c

void error(const char \*msg) {

fprintf(stderr, "Error: %s (%s)\n", msg, strerror(errno));

}

* ****Input Validation:****
  + All scanf statements now check the return value to ensure valid input.
  + flush\_input() function is used to clear invalid input from the buffer.

### 5. User Interface Improvements:

* ****Command-Line Interface Enhancements:****
  + Clear prompts and input validation improve the user experience.
  + Error messages are descriptive, helping users understand what went wrong.
* ****New Commands Added:****
  + alloc**:** Allocate memory.
  + free**:** Deallocate memory.
  + mem**:** Display memory usage.

## How to Compile and Run the Program:

### Step 1: Save the Code:

Save the code in a file named enhanced\_file\_navigation\_system.c.

### Step 2: Compile the Program:

Open a terminal and navigate to the directory containing the code file. Compile the program using GCC with pthread support:

sh

gcc -pthread -o enhanced\_file\_navigation\_system enhanced\_file\_navigation\_system.c

### Step 3: Run the Program:

Execute the compiled program:

sh

./enhanced\_file\_navigation\_system

## Sample Interaction:

Username: user1

Password:

Authentication successful. Welcome, user1!

Enter command (ls, cd, touch, rm, run, alloc, free, mem, exit): ls

enhanced\_file\_navigation\_system.c

other\_files...

Enter command (ls, cd, touch, rm, run, alloc, free, mem, exit): touch

Enter filename to create: example.txt

File 'example.txt' created successfully.

Enter command (ls, cd, touch, rm, run, alloc, free, mem, exit): ls

enhanced\_file\_navigation\_system.c

example.txt

other\_files...

Enter command (ls, cd, touch, rm, run, alloc, free, mem, exit): alloc

Enter memory size to allocate: 256

Allocated 256 units of memory at block 0.

Enter command (ls, cd, touch, rm, run, alloc, free, mem, exit): mem

Memory Blocks:

Block Size Status

0 256 Allocated

Enter command (ls, cd, touch, rm, run, alloc, free, mem, exit): free

Enter memory block number to free: 0

Freed memory block 0.

Enter command (ls, cd, touch, rm, run, alloc, free, mem, exit): mem

Memory Blocks:

Block Size Status

0 256 Free

Enter command (ls, cd, touch, rm, run, alloc, free, mem, exit): run

Scheduling processes using Round Robin algorithm.

Process 1 (Process1) is running.

Process 1 (Process1) has terminated.

Process 2 (Process2) is running.

Process 2 (Process2) has terminated.

Process 3 (Process3) is running.

Process 3 (Process3) has terminated.

Process 4 (Process4) is running.

Process 4 (Process4) has terminated.

Process 5 (Process5) is running.

Process 5 (Process5) has terminated.

Enter command (ls, cd, touch, rm, run, alloc, free, mem, exit): exit

Exiting program.

## Future Scope:

The code we've developed serves as a foundation for exploring and understanding core operating system concepts such as file navigation, process scheduling, memory management, and security mechanisms. There's substantial potential to enhance and expand this program further. Here are some ideas for future development:

### 1. ****Advanced Memory Management:****

* ****Dynamic Memory Allocation Strategies:****
  + Implement algorithms like First Fit, Best Fit, and Worst Fit to manage memory allocation more efficiently.
  + Simulate fragmentation issues and strategies to minimize them.
* ****Paging and Segmentation:****
  + Introduce concepts like paging to divide memory into fixed-sized blocks and manage them effectively.
  + Simulate segmentation to handle memory in variable-sized blocks based on logical divisions.
* ****Virtual Memory:****
  + Implement virtual memory mechanisms to allow processes to use more memory than physically available.
  + Simulate swapping pages in and out of memory, and explore page replacement algorithms like LRU (Least Recently Used).

### 2. ****Enhanced Process Scheduling:****

* ****Additional Scheduling Algorithms:****
  + Implement other algorithms like Priority Scheduling, Shortest Job First (SJF), and Multilevel Queue Scheduling.
  + Allow users to select the scheduling algorithm to use.
* ****Process Control Block (PCB):****
  + Develop a detailed PCB structure to store process-specific information like registers, priority, and I/O status.
  + Use PCBs to manage processes more effectively within the scheduler.
* ****Process Creation and Termination:****
  + Allow users to create custom processes with specified attributes.
  + Implement proper process termination and cleanup routines.
* ****Inter-Process Communication (IPC):****
  + Introduce IPC mechanisms like pipes, message queues, or shared memory.
  + Allow processes to communicate and synchronize with each other.

### 3. ****Improved Security Mechanisms:****

* ****User Authentication and Authorization:****
  + Implement a user management system with multiple user accounts and passwords.
  + Assign roles (e.g., admin, guest) and permissions to restrict access to certain commands or files.
* ****Encryption and Secure Storage:****
  + Use encryption algorithms to securely store sensitive data like passwords.
  + Implement security measures for data transmission between processes or over networks.
* ****Audit Trails and Logging:****
  + Maintain logs of user activities, including login attempts, file access, and command usage.
  + Use logs for monitoring and detecting unauthorized access or unusual activities.

### 4. ****File System Enhancements:****

* ****Hierarchical File System:****
  + Expand file system functionalities to include creating and managing directories and subdirectories.
  + Implement commands like mkdir, rmdir, cp, and mv.
* ****File Permissions and Ownership:****
  + Simulate Unix-like file permissions and ownership.
  + Allow users to change permissions using commands like chmod and chown.
* ****File Metadata Handling:****
  + Display and modify file metadata such as size, creation time, modification time, and access rights.

### 5. ****User Interface Improvements:****

* ****Command-Line Enhancements:****
  + Implement features like command history, auto-completion, and aliases.
  + Provide a help system or manual pages for users to learn about commands (help or man command).
* ****Graphical User Interface (GUI):****
  + Develop a GUI version of the system using libraries like GTK+ or Qt.
  + Provide visual representations of file structures, memory usage, and processes.

### 6. ****Networking Capabilities:****

* ****Remote File Access:****
  + Implement functionalities to access and manage files on remote systems.
  + Use protocols like FTP or SSH for secure file transfer and remote command execution.
* ****Distributed Processing:****
  + Enable processes to be scheduled and executed across multiple networked systems.
  + Explore concepts of distributed computing and load balancing.

### 7. ****Error Handling and Robustness:****

* ****Comprehensive Error Management:****
  + Improve error messages to be more user-friendly and informative.
  + Handle edge cases and unexpected inputs gracefully without crashing.
* ****Input Validation:****
  + Ensure all user inputs are validated to prevent injection attacks or misuse.
  + Implement safeguards against buffer overflows and memory leaks.

### 8. ****Modular and Extensible Design:****

* ****Code Refactoring:****
  + Organize the code into separate modules or classes based on functionality (e.g., file operations, memory management).
  + Increase code reusability and maintainability.
* ****Plugin System:****
  + Design the system to support plugins or extensions, allowing others to add functionalities without modifying the core code.

### **9. **Educational Tools and Documentation:****

* ****Tutorial Mode:****
  + Include an interactive tutorial mode to help new users understand how to use the system and underlying concepts.
* ****Detailed Documentation:****
  + Generate comprehensive documentation for developers, including code comments, design documents, and API references.
* ****Sample Projects:****
  + Provide sample projects or exercises that use the system to teach operating system principles.

### 10. ****Testing and Quality Assurance****

* ****Automated Testing:****
  + Develop unit tests and integration tests to ensure each component functions correctly.
  + Use testing frameworks to automate the testing process.
* ****Continuous Integration/Continuous Deployment (CI/CD):****
  + Set up CI/CD pipelines to automate building, testing, and deploying new versions of the system.
* ****User Feedback Mechanisms:****
  + Implement ways for users to submit feedback, bug reports, or feature requests directly through the system.

### 11. ****Cross-Platform Compatibility:****

* ****Porting to Other Operating Systems:****
  + Modify the code to run on other Unix-like systems or even Windows (using appropriate libraries and system calls).
  + Abstract system-specific functionalities to make the codebase more portable.

### 12. ****Performance Optimization:****

* ****Profiling and Benchmarking:****
  + Use profiling tools to identify performance bottlenecks.
  + Optimize algorithms and data structures for efficiency.
* ****Concurrent Processing:****
  + Enhance the use of multithreading and concurrency to improve the system's responsiveness and performance.

### 13. ****Integration with Real System Components:****

* ****System Call Wrappers:****
  + Provide higher-level abstractions over system calls to simplify development.
  + Ensure safe and controlled access to system resources.
* ****Device Management:****
  + Simulate or interact with hardware devices (e.g., virtual disks, terminals).
  + Explore concepts of device drivers and I/O management.

### 14. ****Artificial Intelligence and Machine Learning Integration:****

* ****Smart Scheduling Algorithms:****
  + Use machine learning techniques to predict process behavior and optimize scheduling decisions.
* ****Anomaly Detection:****
  + Implement AI models to detect unusual patterns in user activity or system behavior for security monitoring.

### 15. ****Collaboration and Open Source Contribution:****

* ****Version Control Systems:****
  + Encourage contributions from the community in the form of code, documentation, or testing.
* ****Community Building:****
  + Create forums, discussion boards, or chat groups for users and developers to interact, share ideas, and provide support.

## **Conclusion:**

We've embarked on an exciting journey to create a comprehensive file navigation system in C, tailored for the Ubuntu terminal. Starting with basic file operations like listing files and changing directories, we've progressively integrated more complex operating system concepts, transforming the program into a rich learning platform.

By adding functionalities such as creating and removing files, we've enhanced the user's ability to interact with the file system dynamically. Introducing advanced process scheduling with a simulated Round Robin algorithm allowed us to explore how operating systems manage multiple processes effectively. The inclusion of process states and burst times provided deeper insight into process lifecycle management.

Our simulated memory management system, with memory allocation and deallocation capabilities, offered a tangible way to understand how memory resources are handled within an operating system. This was further enriched by displaying memory usage, giving a clear picture of resource allocation at any given time.

Security mechanisms were not overlooked. Implementing user authentication emphasized the importance of safeguarding system access and highlighted basic principles of secure programming. Enhancing the user interface with better error handling and input validation made the program more robust and user-friendly, ensuring a smoother interaction and reducing the likelihood of unexpected behavior.

Looking ahead, the future scope of this project is vast and full of potential. There's room to delve into advanced memory management techniques like paging and segmentation, which would simulate how modern operating systems handle memory. Enhancing process scheduling algorithms to include Priority Scheduling or Shortest Job First could provide a more nuanced understanding of process management.

Improving security mechanisms by implementing encryption, user roles, and permissions would not only make the system more secure but also provide practical experience in handling security challenges. Adding networking capabilities could transform the program into a distributed system, opening doors to learn about network protocols and remote interactions.

Integrating a graphical user interface (GUI) would significantly enhance user experience, making the system more accessible and visually engaging. This could be a great opportunity to learn about GUI programming and event-driven design.

In essence, this project is more than just code—it's a stepping stone into the vast world of operating systems. It bridges the gap between theoretical concepts and practical application, allowing for a hands-on approach to learning. By continuing to develop and refine this system, we are not only improving a tool but also expanding our knowledge and skill set in the field of computer science.

Remember, every enhancement is a chance to delve deeper into the intricacies of operating systems. The possibilities are endless, and each new feature brings with it a wealth of learning and discovery.