OS Emulator - Command Line Interface with Marquee

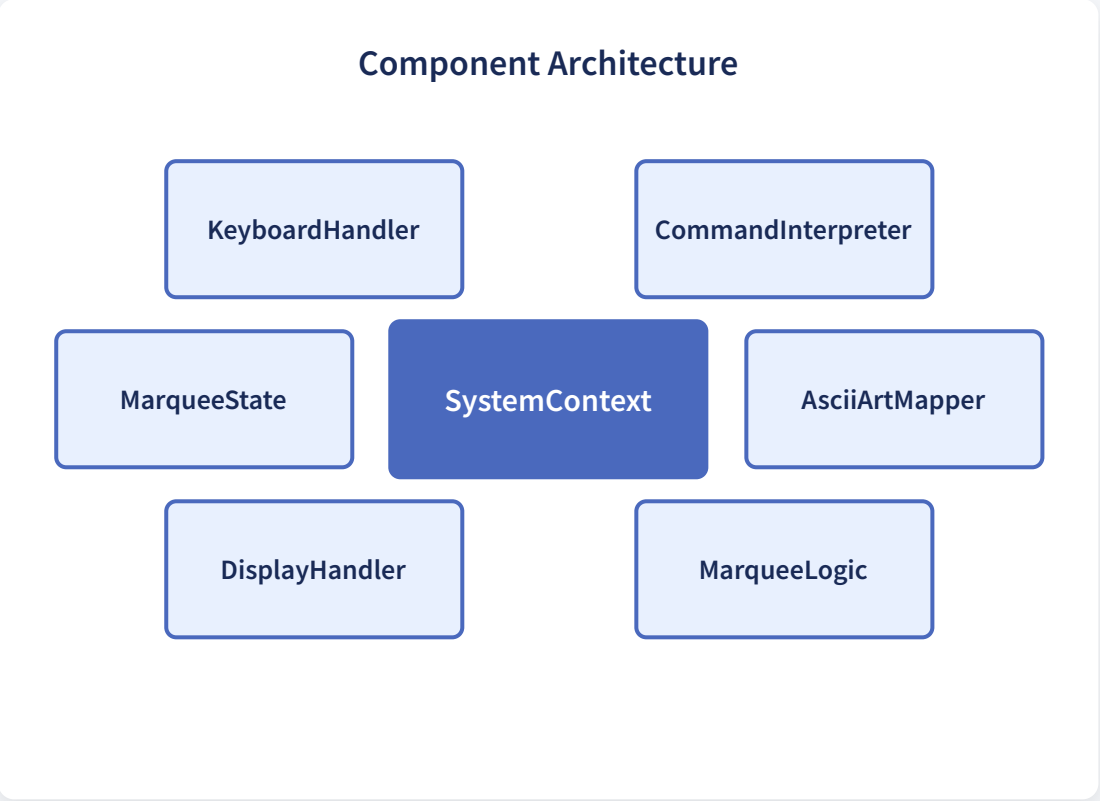
A Technical Implementation Overview

Console UI • Command Interpreter • Display Handler • Marquee Logic

Project Overview

OS Emulator Components

- **SystemContext:** State management
- **KeyboardHandler:** User input
- **CommandInterpreter:** Command execution
- **DisplayHandler:** Screen output
- **MarqueeLogic:** Text animation
- **MarqueeState:** Marquee data
- **AsciiArtMapper:** Text to ASCII



Console UI Implementation

DisplayHandler.cpp - Display Management

```
void DisplayHandler::run() {
    std::cout << "\033[2J\033[H" << "Command > " << std::flush;
    int curr_line = 1;

    while (system_context_ref.is_running) {
        if (system_context_ref.marquee_state.get_active()) {
            if (curr_line < 11) {
                int diff = 11 - curr_line;
                std::cout << "\033[" << diff << "B";
                curr_line = 11;
            }

            std::cout << "\033[s" // save cursor pos
                      << "\033[1;1H" // move cursor top left
                      << marquee_logic_ref.get_next_frame()
                      << "\033[K" // clear to end of line
                      << "\033[u" // restore cursor
                      << std::flush;
        }
        // Handle message display...
        int speed = system_context_ref.marquee_state.get_text_ms();
        std::this_thread::sleep_for(std::chrono::milliseconds(speed));
    }
}
```

main.cpp - Main Application Loop

```
int main() {
    SystemContext context;

    KeyboardHandler keyboard_handler(context);
    CommandInterpreter command_interpreter(context);
    MarqueeLogic marquee_logic(context);
    DisplayHandler display_handler(context, marquee_logic);

    std::thread display_thread(&DisplayHandler::run, &display_handler);

    while (context.is_running) {
        keyboard_handler.run();
        command_interpreter.process_next_command();
        std::this_thread::sleep_for(std::chrono::milliseconds(refresh_rate_ms));
    }

    if (display_thread.joinable()) display_thread.join();
    return 0;
}
```

- **ANSI Escape Codes** for cursor control
- **Multi-threading** for responsive UI
- **Buffer Management** for smooth display
- Refresh rate controlled by **text_ms** parameter
- Separate threads for **display** and **input**

Command Interpreter Implementation

CommandInterpreter.cpp - Command Processing

```
void CommandInterpreter::initialize_commands() {
    commands["help"] = [](const std::string& _) -> std::string {
        return "Commands: help, exit, start_marquee, stop_marquee, set_text , set_speed ";
    };

    commands["exit"] = [this](const std::string& _) -> std::string {
        system_context_ref.is_running = false;
        return "Exiting application...";
    };

    commands["start_marquee"] = [this](const std::string& _) -> std::string {
        system_context_ref.marquee_state.set_active(true);
        return "Marquee started.";
    };

    commands["set_text"] = [this](const std::string& args) -> std::string {
        std::string text = args;
        if (!text.empty() && text[0] == ' ') text.erase(0, 1);
        if (text.empty()) return "Error: No text provided.";

        std::string ascii_art = AsciiArtMapper::to_ascii_art(text);
        system_context_ref.marquee_state.set_text(ascii_art);
        return "Marquee text set to '" + text + "'.";
    };
}
```

CommandInterpreter.cpp - Command Execution

```
std::string CommandInterpreter::execute(const std::string& command_line) {
    std::istringstream iss(command_line);
    std::string command;
    iss >> command;
    std::string args;
    std::getline(iss, args);

    std::string msg;
    auto it = commands.find(command);

    if (it != commands.end()) {
        msg = it->second(args);
    } else {
        msg = "Unknown command: '" + command + "'. Type 'help' for commands.";
    }

    return msg;
}
```

- **Function Map** for command routing
- **Lambda Functions** for command handlers
- **start_marquee:** Activates animation
- **stop_marquee:** Deactivates animation
- **set_text:** Updates marquee content
- **set_speed:** Adjusts refresh rate

Display Implementation

DisplayHandler.cpp - Display Loop

```
void DisplayHandler::run() {
    std::cout << "\033[2J\033[H" << "Command > " << std::flush;
    int curr_line = 1;

    while (system_context_ref.is_running) {
        if (system_context_ref.marquee_state.get_active()) {
            if (curr_line < 11) {
                int diff = 11 - curr_line;
                std::cout << "\033[" << diff << "B";
                curr_line = 11;
            }

            std::cout << "\033[s" // save cursor pos
                      << "\033[1;1H" // move cursor top left
                      << marquee_logic_ref.get_next_frame()
                      << "\033[K" // clear to end of line
                      << "\033[u" // restore cursor
                      << std::flush;
        }

        // Handle message display...
        int speed = system_context_ref.marquee_state.get_text_ms();
        std::this_thread::sleep_for(std::chrono::milliseconds(speed));
    }
}
```

DisplayHandler.cpp - Message Handling

```
std::string msg;
{
    std::lock_guard lock(system_context_ref.prompt_mutex);
    if (!system_context_ref.prompt_display_buffer.empty()) {
        msg = std::move(system_context_ref.prompt_display_buffer);
        system_context_ref.prompt_display_buffer.clear();
    }
}

if (!msg.empty()) {
    std::cout << "\n" << msg << "\nCommand > " << std::flush;
    curr_line += 2;
}
```

- **ANSI Escape Sequences** for terminal control
- **Thread-safe** message buffer management
- **Cursor positioning** with \033[H and \033[u
- **Refresh rate** controlled by marquee speed
- **Separate rendering** for marquee and commands

Marquee Animation Logic

MarqueeLogic.cpp - Frame Generation

```
std::string MarqueeLogic::get_next_frame() {
    MarqueeState state = system_context_ref.marquee_state.get();
    std::string text = state.get_text();

    // split text into lines
    std::vector lines;
    std::istringstream text_stream(text);
    std::string line;
    while (std::getline(text_stream, line)) {
        lines.push_back(line);
    }

    // find width of longest line
    size_t text_width = 0;
    for (const auto& l : lines) {
        text_width = std::max(text_width, l.length());
    }

    const std::string separator = "    ";
    const size_t padded_width = text_width + separator.length();

    // Process each line for animation
    std::ostringstream result_frame;
    for (size_t i = 0; i < lines.size(); ++i) {
        // Create scrolling effect
        std::string loop_line = padded_line;
        while (loop_line.length() < view_width + padded_width) {
            loop_line += padded_line;
        }

        // Get portion for current frame
        std::string frame_part = loop_line.substr(scroll_position % padded_width, view_width);
        result_frame << frame_part;
        if (i < lines.size() - 1) {
            result_frame << "\n";
        }
    }

    // Increment scroll position
    scroll_position++;
    return result_frame.str();
}
```

MarqueeState.cpp - Thread-safe State

```
class MarqueeState {
    std::string text;
    std::mutex mutex;
    int text_ms;
    bool is_active;

public:
    MarqueeState get() {
        std::lock_guard lock(mutex);
        return MarqueeState(text, text_ms, is_active);
    }

    void set_text(const std::string& newText) {
        std::lock_guard lock(mutex);
        text = newText;
    }

    void set_active(bool newActive) {
        std::lock_guard lock(mutex);
        is_active = newActive;
    }

    void set_text_ms(int newTextMs) {
        std::lock_guard lock(mutex);
        text_ms = newTextMs;
    }
};
```

- **Text Buffering** for smooth scrolling
- **Thread Safety** with mutex locks
- **Scroll Position** increments each frame
- **Looping Logic** creates seamless animation
- **Refresh Rate** controlled by text_ms parameter

Conclusion

Key Achievements

Command Line Interface

Fully functional CLI with **6 core commands** for marquee control

Smooth Marquee Animation

Custom ASCII art rendering with **adjustable speed** and text

Modular Architecture

Clean separation of concerns with **7 distinct components**

Technical Challenges Overcome

- **Thread Synchronization** between display and input handlers
- **Performance Optimization** for smooth animation
- **ASCII Art Rendering** with variable character widths
- **Non-blocking Input** while maintaining animation
- **Display Layering** for marquee and command prompt