Final Project Proposal (3-Week Group Project)

Kernel Shell with Memory, Paging, Timer, and System Introspection

1. Introduction

Our group has developed a working instructional kernel featuring paging, a physical page frame allocator, basic device I/O, keyboard input, and VGA text-mode output. However, the current kernel provides no interactive way to inspect or control internal subsystems—any test or debug action requires modifying code and recompiling.

Problem:
The kernel lacks a runtime interface for exploring OS functionality, inspecting memory and paging structures, viewing timekeeping information, or invoking diagnostic tools. This makes development slow and limits the kernel's educational value.

Goal:
We propose implementing an interactive **kernel shell** that operates directly within the OS, enabling real-time introspection of memory, paging, interrupts, and the timer subsystem.

2. System Components & Features

The shell will provide a command-line interface after kernel boot and support several categories of commands:

2.1 Basic Shell Features

- Shell prompt (`> `)
- Line buffer with basic editing (typing, backspace)
- Command parsing into tokens
- Built-in commands:
    - `help` — list all commands
    - `cls` — clear screen
    - `echo <text>` — print text

2.2 Memory / Page-Frame Commands

Using the existing `page.c` allocator:

- meminfo — total vs free frames
- frames — print free frame list
- (Optional, time permitting) alloc <n> and free <addr>

2.3 Paging Commands

Using recursive paging & get_physaddr():

- v2p <virtual_addr> — translate virtual → physical
- ptdump — print PDEs and sample PTEs
- read32 <addr> — read 32-bit word from memory

These commands demonstrate page table layout and kernel mappings.

2.4 Timer Commands

Modify the PIT IRQ0 handler to maintain uptime:

- uptime — display global tick count & time since boot
- (Optional) sleep <ms>

3. Division of Responsibilities (3 Members)

Team Member 1 — Input + Shell Core

- Replace keyboard polling with an interrupt-driven handler
- Implement scancode → ASCII translation
- Implement ring buffer for keystrokes
- Build shell_run() main loop (prompt, line editing, parsing)
- Implement basic commands (help, cls, echo)

Team Member 2 — Memory & Paging Commands

- Implement meminfo, frames, and optional alloc/free
- Implement v2p using get_physaddr()
- Implement ptdump using recursive page tables
- Implement read32 and address parsing

Team Member 3 — Timer + Advanced Commands

- Modify PIT handler to increment a ticks counter
- Add a timer_ticks() API
- Implement uptime command
- Optional stretch commands:
  - info (kernel layout)
  - hexdump <addr> <len>

All members will participate in integration & testing.

4. Project Timeline (3 Weeks)

Week 1 — Input + Shell Skeleton + Basic Commands

Member 1

- Implement IRQ1 keyboard handler (scancode → ASCII → ring buffer)
- Implement keyboard_read_char()
- Build shell loop with prompt and backspace handling
- Implement help, cls, echo

Member 2

- Begin integrating page-frame allocator interface
- Prepare meminfo and frames

Member 3

- Modify PIT handler to maintain a tick counter
- Expose timer_ticks()

Group

- Ensure kernel boots directly into the shell after initialization

Week 2 — Memory & Paging Commands

Member 1

- Improve shell input reliability and parsing
- Add argument parsing (split into argv array)

Member 2

- Finish meminfo, frames
- Implement v2p using get_physaddr()
- Implement read32

Member 3

- Implement uptime
- Begin optional info or hexdump

Group

- Test memory and paging commands in QEMU

Week 3 — Paging Visualization + Final Integration

Member 1

- Polish help messages and command error handling

- Screen clearing, formatting

Member 2

- Implement ptdump using recursive paging:
    - PD at 0xFFFFF000
    - PTs at 0xFFC00000 + index*0x1000

Member 3

- Add optional sleep, idtdump, or hexdump
- Prepare demo + documentation

Group

- Full integration testing
- Final documentation
- Demo script preparation

5. Deliverables

Source Code

- shell.c, shell.h — shell engine
- Updated keyboard ISR (IRQ1) & ring buffer
- Updated PIT handler (IRQ0)
- Memory and paging command implementations
- Integration with current kernel boot path

Documentation

- Shell architecture overview

- All commands + example usage
- Recursive paging explanation
- Timer uptime explanation
- Known limitations & future improvements

Demonstration

In QEMU:

- Show the shell prompt
- Run:
    - help, cls, echo
    - meminfo, frames
    - v2p, ptdump
    - uptime

6. Repository & Collaboration Plan

All work will be done in:

https://github.com/BlakeBrenner/OS-final-project

Branches:

- shell-core
- memory-paging-commands
- timer-commands
  Merged into:

main

Weekly team sync to integrate subsystems.

7. Conclusion

This 3-week project will produce an interactive kernel shell that ties together multiple OS subsystems (interrupts, memory allocation, paging, timer hardware, VGA output) into a cohesive tool.

The project is:

- technically meaningful,
- scoped well for a 3-week timeline,
- divided cleanly across 3 team members,
- and results in a lasting debugging tool for future work.