SYS-101:00020 (Fall 2025)
Kyaw Zin Htet (Ryan)

Bare-Metal Shell — Design Document

Purpose
A `#[no_std]` x86_64 kernel that boots via UEFI and runs a small interactive shell on the QEMU framebuffer. Keyboard input and a bump allocator enable the shell to function.

How the software interacts with kernel parts
Boot flow:

- Host launcher (`src/main.rs`) builds UEFI image (`build.rs` sets `UEFI_PATH`), downloads OVMF, and runs `qemu-system-x86_64`.
- Bootloader enters the kernel at `kernel/src/main.rs::kernel_main`.

Subsystems initialization in `kernel_main`:

- Framebuffer console: `kernel/src/screen.rs::init` creates a global `ScreenWriter` used via `Writer` (implements `core::fmt::Write`).
- Memory/heap: scans `BootInfo` memory regions, computes a virtual heap base using `physical_memory_offset`, calls `allocator::init_heap`.
- GDT/TSS: `kernel/src/gdt.rs::init` for reliable interrupts.
- APIC/IDT: `kernel/src/interrupts.rs` parses ACPI, maps LAPIC/IOAPIC, installs IDT entries, enables interrupts.
- Handlers wiring: `HandlerTable` (in `kernel/src/lib.rs`) registers `startup`, `timer`, and `keyboard` callbacks, then starts the system.

Kernel design (high level)
Modules:

- `screen.rs`: Framebuffer text rendering (glyph raster via `noto_sans_mono_bitmap`), clear, newline, carriage-return, pixel drawing.
- `allocator.rs`: Global bump allocator; `alloc` aligns and bumps; `dealloc` no-op; `memstat` reports usage.
- `interrupts.rs`: ACPI/APIC setup, IDT, timer + keyboard ISRs, EOI; forwards to `HandlerTable`.
- `main.rs`: Shell struct and logic; connects handlers; top-level boot orchestration.
- `lib.rs`: Serial logging (`uart_16550`), panic handler, `HandlerTable`, `hlt_loop`.

Memory allocator

- Location: `kernel/src/allocator.rs`.
- Type: Simple bump allocator with `HEAP_START`, `OFFSET`, fixed `HEAP_SIZE` (100 KiB).
- Init: `allocator::init_heap(virtual_heap_base)` sets the heap start and resets `OFFSET`.
- Alloc: Aligns upward from `HEAP_START + OFFSET`, bumps `OFFSET`, returns pointer; OOM returns null (logged via serial).
- Dealloc: No-op (acceptable for this assignment).
- Reporting: `allocator::memstat() → (used, total)` backs the `memstat` command.

Input controller (keyboard)

- Hardware: Keyboard scancodes read from port 0x60 in `keyboard_interrupt_handler` (`interrupts.rs`).
- Decoding: `pc_keyboard` converts scancodes to `DecodedKey`.
- Dispatch: ISR calls `HandlerTable.keyboard(key)`; kernel wires this to `Shell::handle_key`.
- Shell behavior:
    - Printable chars: appended to buffer, echoed to screen using `Writer`.
    - Enter: executes the buffer (parsing + dispatch), clears buffer, prints new prompt.
    - Backspace: removes last char and redraws the line (prompt + content).

State management

- Shell state: `lazy_static! static ref SHELL: Mutex<Shell>` with `buf: String`, `ticks: u64`.
- Screen writer: global `RacyCell<Option<ScreenWriter>>` initialized once at boot; accessed through `Writer` adapter.
- Timer tick: APIC timer ISR calls `HandlerTable.timer()`, which increments `SHELL.ticks`.
- Concurrency: Single-core, interrupt-driven; shared state guarded by `spin::Mutex`.

Command parsing and dispatcher

- Parsing: `split_whitespace`; first token is command, remainder are args.
- Dispatcher: `match` on command string.
- Built-ins:
    - `echo [text...]` — prints args joined by space
    - `clear` — clears framebuffer, resets cursor
    - `ticks` — shows number of timer interrupts since boot
    - `memstat` — shows `(used / total bytes)` from allocator
    - `help` — lists available commands
    - Unknown — prints `unknown: <cmd>`

Additional features

- Colored pixel bars drawn at boot to validate framebuffer path.
- Serial logs for debugging (panic handler and init stages), while all user-visible output goes to framebuffer.

This is github repository - https://github.com/Ryan-100/sys-101-00020