The UART Module

This chapter introduces the UART (Universal Asynchronous Receiver-Transmitter) module of Novix OS.
UART is the primary way for the kernel to send and receive data to and from the host — it is our basic input/output device.
It is also used extensively throughout debugging, logging, and communication between the OS and QEMU console.

Function: uart_getc

```c
long uart_getc(void);
```

Description:
Performs an SBI call to receive a single character from the UART input buffer.
Returns `-1` if no character is available.

Function: uart_putc

```c
void uart_putc(char ch);
```

Description:
Sends a single character to the UART output using SBI call (extension ID `1`).

Function: uart_puts

```c
void uart_puts(const char *s);
```

Description:
Prints an entire string by repeatedly calling `uart_putc()` until a null terminator is reached.

Function: uart_puthex8 / uart_puthex8_prefixed

```c
void uart_puthex8(uint8_t val);
void uart_puthex8_prefixed(uint8_t val);
```

Description:
Prints an 8-bit value in hexadecimal; the prefixed version adds `"0x"`.

Function: uart_puthex16 / uart_puthex16_prefixed

```c
void uart_puthex16(uint16_t val);
void uart_puthex16_prefixed(uint16_t val);
```

Description:
Prints a 16-bit value in hexadecimal; the prefixed version adds `"0x"`.

Function: uart_puthex32 / uart_puthex32_prefixed

```c
void uart_puthex32(uint32_t val);
void uart_puthex32_prefixed(uint32_t val);
```

Description:
Outputs a 32-bit value in hexadecimal format. The prefixed version includes `"0x"`.

Function: uart_puthex64 / uart_puthex64_prefixed

```c
void uart_puthex64(uint64_t val);
void uart_puthex64_prefixed(uint64_t val);
```

Description:
Prints a 64-bit value in hexadecimal; `_prefixed` adds `"0x"`. Used for printing addresses and large integers.

Function: uart_put_udec

```c
void uart_put_udec(uint64_t value);
```

Description:
Prints an unsigned integer in decimal format. Implemented manually without libc.

Function: uart_put_dec

```c
void uart_put_dec(int64_t value);
```

Description:
Prints a signed integer with a leading minus sign if negative.

Function: uart_puthex

```c
void uart_puthex(uint64_t value, int width);
```

Description:
Prints any integer value in hexadecimal format, padding zeros to match width.

Function: uart_putpad

```c
void uart_putpad(uint64_t val, int base, int width, int zero_pad, int signed_val);
```

Description:
Flexible number formatting helper that supports padding, base, and signed output.

Function: uart_printf

```c
void uart_printf(const char *fmt, ...);
```

Description:
Implements a minimal formatted print system supporting `%d`, `%u`, `%x`, `%p`, `%c`, `%s`, and padding options.

Function: uart_cls

```c
void uart_cls(void);
```

Description:
Clears the terminal screen using ANSI escape sequences.

Function: uart_set_color

```c
void uart_set_color(const char *ansi_code);
```

Description:
Sets ANSI color for terminal text (e.g. `"\x1b[31m"` for red).

Test Functions

Function: heap_test

Description:
Tests dynamic memory allocation by allocating and freeing several memory blocks.

Function: getc_test

Description:
Waits for a character input via UART, then echoes it to confirm UART RX/TX.

Function: date_time_test

Description:
Fetches and prints current time using `get_time()` and `compute_datetime()`.

Function: log_test

Description:
Demonstrates `LOG_INFO`, `LOG_WARN`, `LOG_ERR`, and `LOG_DEBUG` macros for kernel logging.

Function: kernel_main

Prototype

```c
void kernel_main(void);
```

Parameters

- None.

Return Value

- None.
  This function does not return — it contains an infinite loop to keep the kernel active.

Purpose

`kernel_main()` is the main entry point of the Novix OS kernel after the bootloader (`boot()` function) has set up the stack and transferred control.
It initializes the system's basic runtime environment and provides user feedback through UART output.

Detailed Description

1. **Screen Initialization**
   The function starts by clearing the UART terminal screen using:

   ```c
   uart_cls();
   ```

   This ensures that the kernel output starts from a clean display after boot.

2. **Welcome Message**
   Next, a versioned welcome message is printed to the console:

   ```c
   uart_printf("Novix RISC-V 64 OS, (c) NovixManiac, Version 0.0.1\n\n");
   ```

   This identifies the running OS and confirms successful control transfer from the bootloader.

3. **Boot Feedback**
   The message `"Booting ..."` is printed using:

   ```
   uart_puts("Booting ...\n");
   ```

   This provides immediate user feedback that the kernel startup process has begun.

4. **UART Input Test**
   The kernel performs a basic UART input test by calling:

   ```
   getc_test();
   ```

   This test prompts the user to type a key and echoes it back to verify both UART transmission and reception functionality.

5. **Infinite Loop**
   Finally, the kernel enters an infinite loop:

   ```
   for (;;);
   ```

   This keeps the kernel running indefinitely, preventing control from returning to the bootloader and maintaining the system in an operational state.

Summary

The `kernel_main()` function is the central initialization point for the Novix kernel.
It demonstrates successful boot, performs a UART diagnostic, and transitions the system into its idle operational state.