Ethernet, TCP klient a server

1 Zadání

- Připojte vývojovou desku NUCLEO-F429ZI do lokální sítě, použijte FreeRTOS a lwIP. Vyzkoušejte ping desky. Implementujte tcpecho využívající netconn API **(1b)**.
- Podle vzoru tcpecho vytvořte nový thread, zajišťující telnet server na portu 23. Implementujte ovládání LD1 až LD3 pomocí textového protokolu podobného jeho v cvičení *UART komunikace s DMA, EEPROM na I2C* **(2b)**.
- Aktivujte HTTP server pro raw API, vytvořte základní souborový systém webových stránek a ověřte funkci **(1b)**.

Příkazy textového protokolu:

- HELLO
- LED1 ON|OFF
- LED2 ON|OFF
- LED3 ON|OFF
- STATUS

2 Návod

2.1 Základní seznámení

- Vytvořte si pracovní kopii svého repozitáře z Githubu (Git Clone), příp. aktualizujte repozitář (Git Pull).
- Založte nový projekt přes File / New / STM32 Project / Board Selector / NUCLEO-F429ZI. Budeme využívat HAL knihovny, proto ponechte Targeted Project Type na STM32Cube. Potvrďte inicializaci všech periferií do výchozího nastavení.
- Je vhodné deaktivovat nepoužívané periferie, tj. vypnout USB_OTG_FS (Mode: Disable), aby projekt nebyl zbytečně komplikovaný.

2.2 Konfigurace ethernetu a ping

- Budeme využívat knihovnu lwIP pod systémem FreeRTOS. V CubeMX je třeba povolit FreeRTOS (CMSIS_V1) a podle doporučení přepnout časovou základnu (SYS / Timebase Source = TIM14). Veškeré úlohy budou vytvářeny dynamicky, v konfiguraci FreeRTOS pod CubeMX je tedy nevytváříme.
- Vzhledem k rozsahu projektu je potřeba zvětšit velikost haldy FreeRTOS (FREERTOS / Config Parameters / Memory management settings / TOTAL_HEAL_SIZE = 32768). Dále je vhodné zvětšit velikost zásobníku defaultTasku, v rámci kterého jsou prováděny inicializace (FREERTOS / Tasks and Queues / defaultTask / Stack Size = 1024).
- Výchozí nastavení ethernetu je správné s výjimkou MAC adresy, ta musí být pro každou vývojovou desku jedinečná a zaregistrovaná v síti FEKT. Zvolte v ETH / Parameter Settings / Ethernet MAC Address adresu **00:80:E1:FE:EC:nn**, kde **nn** je číslo vašeho PC. Např. pro PC-071 tedy bude adresa 00:80:E1:FE:EC:71. Tomu odpovídá DNS adresa VD-STM-071.urel.feec.vutbr.cz.
- Aktivujte lwIP (Enabled), vygenerujte kód, přeložte a spusťte.
- Po připojení do sítě se rozsvítí LED kontroly na ethernetovém konektoru, ověřte funkčnost pomocí příkazu ping na adresu VD-STM-0**nn**.urel.feec.vutbr.cz.

- Získanou IP adresu lze ověřit pauznutím běhu programu a přidáním symbolu gnetif do okna Expressions. Po rozbalení gnetif / ip_addr / addr byste měli najít nenulovou IP adresu přiřazenou DHCP serverem v decimálním vyjádření.

2.3 Příklad tcpecho

- Příklad tcpecho (viz eLearning) je součástí balíku lwIP. Poslouchá na TCP portu 7 a veškerá přijatá data odesílá zpátky. Použitá verze je postavena na netconn API.
- V knihovně lwIP jsou podporována tři základní API (viz https://www.nongnu.org/lwip/2_0_x/raw_api.html):
  - Raw API (někdy se nazývá nativní nebo callback API) je navrženo pro použití bez operačního systému, implementuje zero-copy odesílání a příjem. Toto API také používá jádro lwIP pro interakci mezi různými protokoly. Je to jediné API dostupné při spuštění lwIP bez operačního systému. Použití je komplexní.
  - Sekvenční API (netconn API) poskytuje způsob pro běžné, sekvenční programy. Je podobné BSD socket API. Model běhu je založen na blokujících voláních open-read-write-close. Kód pro TCP/IP a aplikační programy musí být umístěn v rozdílných vláknech. Je programátorsky přívětivé. Obvyklá volba pro vlastní implementace.
  - Socket API je kompatibilní s BSD, je postaveno na netconn API. Je velice rozsáhlé, vzhledem ke své univerzálnosti zabírá hodně paměti RAM i kódu.
- Soubor tcpecho.c vložte mezi zdrojové kódy projektu (Core/Src). V souboru main.c deklarujte prototyp inicializační funkce:

```
extern void tcpecho_init(void);
```

- Tuto funkci zavolejte v rámci inicializace (tj. před nekonečnou smyčkou) ve StartDefaultTask:

```
/* Initialize tcp echo server */
tcpecho_init();
```

- Ověřte pomocí PuTTY připojení na port 7, veškeré odeslané zprávy bude tcpecho server vracet zpět.

2.4 Telnet server

- Pro implementaci telnet serveru vyjdeme z tcpecho příkladu. Soubor tcpecho.c okopírujte pro projektu jako telnet.c, přejmenujte příslušné funkce (veškeré odkazy na tcpecho nahraďte za telnet) a změňte port na 23.
- Inicializaci do main.c doplníme stejným způsobem jako u tcpecho serveru. Projekt nyní obsahuje dva servery – tcpecho (port 7) a telnet (port 23 – zatím pouze s echo funkcí). Oba servery běží současně.
- Dále budeme implementovat příkazy pro práci s LED viz zadání. Použijeme stejný postup jako ve cvičení *UART komunikace s DMA, EEPROM na I2C*, tj. budeme znaky zpracovávat jeden po druhém, ignorovat neplatné a na závěr ze složených řetězců parsovat příkazy. Nově je třeba předávat identifikátor spojení (struct netconn *), do kterého budeme odpovídat.
- Ve vláknu telnet_thread() budeme přijímat data typu uint8_t *data, vnitřní smyčku do {} while tedy přepíšeme například takto:

```
netbuf_data(buf, (void**)&data, &len);
while (len--) telnet_byte_available(*data++, newconn);
```

- Funkce telnet_byte_available() bude reagovat na jednotlivé přijaté bajty a skládat je do pole:

```
static void telnet_byte_available(uint8_t c, struct netconn *conn)
{
    static uint16_t cnt;
    static char data[CMD_BUFFER_LEN];

    if (cnt < CMD_BUFFER_LEN && c >= 32 && c <= 127) data[cnt++] = c;
    if (c == '\n' || c == '\r') {
        data[cnt] = '\0';
        telnet_process_command(data, conn);
        cnt = 0;
    }
}
```

- Nakonec je volána funkce telnet_process_command(), která provede parsování a vykonání povelů:

```
static void telnet_process_command(char *cmd, struct netconn *conn);
```

- Není možné používat printf() pro standardní výstup, odpovědi je třeba vypisovat pomocí sprintf() do textového řetězce a ten následně odeslat pomocí netconn volání:

```
netconn_write(conn, s, strlen(s), NETCONN_COPY);
```

- Funkce strtok() není reentrantní a nelze tedy snadno použít ve vícevláknovém prostředí FreeRTOS (pokus skončí hardfaultem). Řešením je použít upravenou funkci strtok_r(), která využívá pomocný lokální parametr char *saveptr, viz např. https://linux.die.net/man/3/strtok_r.
- Po připojení telnetem na vývojovu desku ověřte funkci pomocí v zadání uvedených příkazů.

2.5 HTTP server

- V konfiguraci lwIP aktivujte HTTP server (LWIP / HTTPD / LWIP_HTTPD = Enabled).
- Tento poměrně komplexní HTTP server je postavený na raw API. Webové stránky v nejjednodušším případě čte z paměti programu, kam jsou vloženy pomocí souboru fsdata_custom.c.
- Pomocí přiloženého programu makefsdata.exe (zdrojový kód je součástí balíku lwIP) zkompilujte HTML soubory a obrázky ve složce Fs do souboru fsdata.c. Tento soubor přejmenujte na fsdata_custom.c, vložte ho mezi **hlavičkové** soubory projektu (Core/**Inc**) a vyřaďte z kompilovaných souborů (pravé tlačítko pro kontextové menu / Resource Configurations / Exclude from Build).
- Tento postup řeší situaci, kdy soubor chceme mít přístupný v IDE, ale nekompilujeme ho samostatně – je totiž pomocí #include vložený do souboru Middlewares/Third_Party/LwIP/src/apps/http/fs.c. Musí tedy být v prohledávané cestě, aby fungoval #include (proto Core/Inc), a zároveň nesmí být samostatně kompilován (jinak by symboly v něm uvedené byly definované vícenásobně).
- V souboru main.c includujte příslušný hlavičkový soubor:

```
#include "lwip/apps/httpd.h"
```

- A ve StartDefaultTasku inicializujte HTTP server:

```
/* Initialize HTTP server */
httpd_init();
```

- Projekt obsahuje tři servery – tcpecho (port 7), telnet (port 23) a HTTP (port 80). Všechny servery běží současně.
- Ověřte funkci pomocí webového prohlížeče na adrese http://VD-STM-0**nn**.urel.feec.vutbr.cz.