VIN Projekt 2023/2024

UART ukazna vrstica za USB datotečno napravo na plošči STM32H750B-DK

Avtor: Tilen Bezgovšek

17.6.2024

Uvod

Cilj projekta je ustvariti uporabniški vmesnik za upravljanje z datotekami na USB napravi preko ukazne vrstice. Sporočila se prenašajo preko protokola UART.

Najprej nekaj osnovnih zahtev za izvedbo projekta. USB s ploščo povežemo preko porta USB OTG (on the go).

Z USB OTG kablom povežemo USB napravo s ploščo in preko porta STLink ploščo povežemo z računalnikom.

Za delo bomo uporabili knjižnico FatFs, ki omogoča enostavno delo s FAT datotečnim sistemom. Zato mora biti naprava formatirana v datotečnem sistemu FAT32.

Kreiranje projekta s CubeMX

Jaz sem za delo projekta uporabljal IDE program STM32CubeIDE.

Ustvariti je treba nov STM32 projekt s CubeMX grafičnim vmesnikom in vklopiti naprave, ki jih potrebujemo.

Potrebne nastavitve za projekt

USART3

- Mode: Asynchronous

- ostalo pustimo privzeto (Baud Rate je 115200 bps)

USB_OTG_FS

- Mode: Host Only – za ploščo izberemo način gostitelja

- Activate VBUS: VBUS sensing

USB_HOST

- Class for FS IP: Mass Storage Host Class

- Platform Settings: tu vidimo, da je Drive_VBUS_FS “undefined” in “No solution”. Drive_VBUS je napajanje USB naprave preko USB OTG porta in za naprave, ki nimajo svojega napajanja (na primer USB ključ) mora biti napajanje omogočeno. Zato je treba nastaviti pravilni pin.

GPIO

V navodilih za uporabo za ploščo lahko vidimo, da je pin PA5 (T3): OTG_FS2_PSO. “PSO” je “power supply output”. Ta pin bomo uporabili za napajanje. V Pinout View pin PA5 (T3) nastavimo na GPIO_Output in nastavimo GPIO output level na “high”.

USB_HOST

- Platform Settings: zdaj lahko za Drive_VBUS_FS izberemo GPIO:Output in izberemo solution PA5.

FATFS

- Mode: USB Disk

- Configuration: USE_LFN: Enabled with static working buffer on the BSS. FS_RPATH: Enabled with f_gcwd. MAX_SS: 4096.

Zdaj je vse pripravljeno. Shranimo projekt in generiramo kodo.

Priprava za delo s FatFs

V generirani kodi lahko v datoteki main.c vidimo, da je v glavno while zanko dodan klic funkcije MX_USB_HOST_Process. Pred tem pa so dodane tudi za nas pomembne inicializacijske funkcije:

MX_USART3_UART_Init, MX_USB_HOST_Init in MX_FATFS_Init.

UART komunikacija

Za pošiljanje sporočil med ploščo in ukaznim oknom bomo uporabili UART. Za to lahko uporabimo program PuTTY. Hitrost se mora ujemati z nastavitvami, v našem primeru je to 115200.

Osnovni funkciji, ki ju bomo uporabljali sta HAL_UART_Transmit in HAL_UART_Receive. Pri obeh je prvi argument kazalec na UART objekt, definiran v main.c (v našem primeru huart3), drugi kazalec na buffer s podatki (po domače tabela/polje, kjer so/bodo podatki shranjeni), velikost (število poslanih elementov) in timeout.

USB naprava

Če pogledamo še inicializacijsko funkcijo MX_FATFS_Init, vidimo da se nahaja vse, kar je v njej, klic funkcije FATFS_LinkDriver. Ta funkcija poveže FatFs z USB diskom. Mi moramo pa samo poskrbeti, da na koncu našega programa kličemo funkcijo FATFS_UnLinkDriver, ki prekine povezavo.

Zdaj lahko pogledamo še funkcijo MX_USB_HOST_Process v glavni while zanki. Ta se nahaja v datoteki usb_host.c in vsebuje vse za inicializacijo USB naprave in za delo s to napravo.

Za nas je najbolj pomembna funkcija USBH_UserProcess, ki je v bistvu pripravljena za klicanje novo ustvarjenih uporabniških funkcij.

UART input

Za ukazno vrstico je treba napisati še funkcijo za branje vhoda iz tipkovnice. Za delo z nizi bomo potrebovali tudi knjižnico string.h.

```
char* read_input() {
	char* str = (char*)malloc(BUFSIZE * sizeof(char));
	memset(str, 0, BUFSIZE);
	size_t len = 0;
	char ch;

	do {
		HAL_UART_Receive(&huart3, (uint8_t *)&ch, 1, HAL_MAX_DELAY);
		if (ch != '\n' && ch != '\r') {
			if (ch != '\177') {
				str[len] = ch;
				len++;
			} else if (len > 0) {
				len--;
			}
			HAL_UART_Transmit(&huart3, (uint8_t *)&ch, 1, 100);
		} else {
			break;
		}
	} while (len < BUFSIZE - 1);

	HAL_UART_Transmit(&huart3, (uint8_t *)nl, strlen(nl), 100);
	return str;
}
```

To je enostavno branje tipkovnice znak po znaku, shranjevanje znaka v buffer in pošiljanje prejetega znaka nazaj, da lahko dejansko vidimo, kaj smo napisali.

Edina posebnost tu je znak za “delete”. Če bi brez izjem vsak znak takoj shranili, brisanje ne bi delovalo. Zato je treba najprej preveriti, če je bila pritisnjena tipka za brisanje. V tem primeru je treba naš indeks prestaviti nazaj, da naslednji vnešeni znak povozi prejšnjega.

Delo s FatFs

Funkcija USBH_UserProcess izgleda takole:

```
static void USBH_UserProcess  (USBH_HandleTypeDef *phost, uint8_t id)
{
  /* USER CODE BEGIN CALL_BACK_1 */
  switch(id)
  {
  case HOST_USER_SELECT_CONFIGURATION:
  break;

  case HOST_USER_DISCONNECTION:
  Appli_state = APPLICATION_DISCONNECT;
  break;

  case HOST_USER_CLASS_ACTIVE:
  Appli_state = APPLICATION_READY;
  break;

  case HOST_USER_CONNECTION:
  Appli_state = APPLICATION_START;
  break;

  default:
  break;
  }
  /* USER CODE END CALL_BACK_1 */
}
```

Mi bi radi naše funkcije klicali takrat, ko je USB naprava pripravljena za delovanje. To je `case HOST_USER_CLASS_ACTIVE`. Zato bomo v ta case dodali našo kodo.

Za samo delo s funkcijami FatFs priporočam ogled dokumentacije, ki zelo dobro pojasni delovanje:

https://elm-chan.org/fsw/ff/00index_e.html

Če link ne dela, je dostopna tudi na:
https://dev.ti.com/tirex/explore/content/simplelink_msp432e4_sdk_4_20_00_12/source/third_party/fatfs/documents/00index_e.html

FatFs mount in unmount

Najprej je treba datotečni sistem naložiti ali montirati (ang. mount) in na koncu ga je treba odmontirati (ang. unmount).

Za to uporabimo funkcijo f_mount. Pred tem je treba še inicializirati spremenljivko tipa FATFS oziroma kazalec na to spremenljivko. Podati je treba še številko pogona (ang. drive). Če imamo enega samega, lahko podamo prazen niz.

Tu bi rad poudaril še pomembnost preverjanja rezultatov klica FatFs funkcij. Vsaka FatFs funkcija vrača rezultat tipa FRESULT, ki nam pove, če se je vse izvedlo po pričakovanjih. Če je vse v redu, je rezultat FR_OK, v primeru kakršnihkoli težav pa je rezultat specifičen za vrsto napake. Pri vsakem klicu funkcij je priporočljivo preverjanje pravilnosti delovanja.

Za odmontiranje lahko kličemo isto funkcijo f_mount z NULL kot prvi argument. Po odmontiranju je treba dodati še klic funkcije FATFS_UnLinkDriver, ki sem ga omenil prej.

```
void mount_FatFs() {
       char mErr[] = "Mount error!";
       FRESULT res;

       UsbDiskFatFs = malloc(sizeof(FATFS));

    res = f_mount(UsbDiskFatFs, (const TCHAR*)UsbDiskPath, 0);
    if (res != FR_OK) {
       /* FatFs Init Error */
       HAL_UART_Transmit(&huart3, (uint8_t *)mErr, strlen(mErr), 100);
       HAL_UART_Transmit(&huart3, (uint8_t *)nl, strlen(nl), 100);
       while(1);
    } else {
       mounted = 1;
    }
}

void unmount_FatFs() {
       /* Unmount the device */
       f_mount(NULL, UsbDiskPath, 0);
       /* Unlink the USB disk driver */
       FATFS_UnLinkDriver(UsbDiskPath);
}
```

Funkcija parse_input

Glavna funkcija je parse_input. Funkcija kot vhod dobi niz znakov, ki predstavlja uporabnikov vnos. Najprej se vhod s pomočjo funkcije strtok razdeli na besede. Nato se glede na prvo besedo izbere ukaz. Funkcija vrača rezultat tipa FRESULT in ponavadi kar posreduje rezultat relevantne FATFS funkcije.

```
FRESULT parse_input(char* str) {
	FRESULT res;
	char unknown[] = "Unknown command!";
	char exit[] = "Exiting...";
	char* tokens[3] = {0};
	char* delims = " \t";
	char* token = strtok(str, delims);
	for (int i = 0; i < 3; i++) {
		if (token == NULL) {
			break;
		}
		tokens[i] = token;
		token = strtok(NULL, delims);
	}
	…
}
```

Ukaz pwd

To je verjetno najbolj enostaven ukaz za implementacijo. Ker smo pri pripravi projekta v CubeMX za FatFs omogočili uporabo relativnih poti, lahko uporabimo funkcijo f_getcwd. Ta funkcija vrne trenutno pot. Če slučajno tega ne bi takrat naredili, lahko še vedno nastavitve FatFs popravimo tako, da odpremo datoteko ffconf.h in _FS_RPATH nastavimo na 2.

Za ukaz pwd samo kličemo funkcijo f_getcwd z izbranim bufferjem in preverimo rezultat. Nato preko UART pošljemo dobljeno pot.

```
TCHAR path[PATHSIZE] = {0};
res = f_getcwd(path, PATHSIZE);
if (res != FR_OK) {
	return res;
}
HAL_UART_Transmit(&huart3, (uint8_t *)path, strlen(path), 100);
HAL_UART_Transmit(&huart3, (uint8_t *)nl, strlen(nl), 100);
```

Ukaz ls

Za ukaz ls moramo preveriti, če je bila podana druga beseda pri uporabniškem vnosu. Če je ni, bomo izpisali seznam za trenutno pot. Če pa je, pa poskusimo izpisati seznam elementov za podano pot.

Tu lahko uporabimo funkcijo f_findfirst. Potrebujemo kazalca na spremenljivko tipa DIR za direktorij in FILINFO za podatke o najdenem objektu. Funkciji podamo pot in funkcija nam direktorij odpre ter postavi kazalec na podatke za prvi objekt. Za naslednje objekte v direktoriju lahko uporabimo funkcijo f_findnext. Podamo ji odprt direktorij in kazalec na FILINFO spremenljivko.

Za vsak objekt preverimo, če je direktorij in nato izpišemo izbrane podatke (na primer ime, velikost, tip objekta).

Na koncu samo še zapremo direktorij.

```
DIR dir;
FILINFO finfo;
TCHAR path[PATHSIZE] = {0};
if (tokens[1] != NULL) {
	strncpy(path, tokens[1], sizeof(path) - 1);
} else {
	res = f_getcwd(path, PATHSIZE);
	if (res != FR_OK) {
		return res;
	}
}
res = f_findfirst(&dir, &finfo, path, "*");
if (res != FR_OK) {
	return res;
}
do {
	if (res == FR_OK && finfo.fname[0]) {
		if (finfo.fattrib & AM_DIR) {
			snprintf(toPrint, BUFSIZE, "%-5s%20s/%-14s\n\r", "DIR", path, finfo.fname);
		} else {
			snprintf(toPrint, BUFSIZE, "%-5s%20s/%-14s%10li B\n\r", "FILE", path,
finfo.fname, finfo.fsize);
		}
		HAL_UART_Transmit(&huart3, (uint8_t *)toPrint, strlen(toPrint), 100);
	}
	res = f_findnext(&dir, &finfo);
} while (res == FR_OK && finfo.fname[0]);
f_closedir(&dir);
```

Ukaz cd

Za ukaz cd lahko uporabimo funkcijo f_chdir. Tako spremenimo trenutno pot.

```
TCHAR path[PATHSIZE] = {0};
if (tokens[1] != NULL) {
	strncpy(path, tokens[1], sizeof(path) - 1);
} else {
	strncpy(path, "/", sizeof(path) - 1);
}
res = f_chdir(path);
if (res != FR_OK) {
	return res;
}
```

Ukaz read

Za ukaz read bomo uporabili funkcijo f_open. Ta funkcija nam datoteko odpre na podan način. Kot zadnji argument je treba podati “mode” oziroma način. To je način dostopa do datoteke.

Flags	Meaning
FA_READ	Specifies read access to the object. Data can be read from the file.
FA_WRITE	Specifies write access to the object. Data can be written to the file. Combine with FA_READ for read-write access.
FA_OPEN_EXISTING	Opens the file. The function fails if the file is not existing. (Default)
FA_CREATE_NEW	Creates a new file. The function fails with FR_EXIST if the file is existing.
FA_CREATE_ALWAYS	Creates a new file. If the file is existing, it will be truncated and overwritten.
FA_OPEN_ALWAYS	Opens the file if it is existing. If not, a new file will be created.
FA_OPEN_APPEND	Same as FA_OPEN_ALWAYS except the read/write pointer is set end of the file.

Te načine lahko tudi kombiniramo z logičnim OR operatorjem (“|”).

Ko uporabljamo funkcij f_open, je smiselno bolj podrobno razdelati, če je funkcija neuspešna. Če na primer vrne FR_NO_FILE pomeni, da datoteke ni našla. FR_NO_PATH pomeni, da poti ni našla. Za uporabnika so to uporabne informacije.

Ko imamo datoteko odprto, uporabimo funkcijo f_gets. Ta nam iz tekstovne datoteke prebere in shrani tekst. V zanki beremo v buffer, dokler ne dobimo rezultat funkcije 0.

Na koncu obvezno datoteko zapremo z f_close.

```
FIL file;
char line[100] = {0};
char* input;
if (tokens[1] == NULL) {
	HAL_UART_Transmit(&huart3, (uint8_t *)"Please provide filename.", 24, 100);
	HAL_UART_Transmit(&huart3, (uint8_t *)nl, strlen(nl), 100);
} else {
	res = f_open(&file, tokens[1], FA_READ);
	if (res == FR_NO_FILE) {
		HAL_UART_Transmit(&huart3, (uint8_t *)"Could not find the file. Create file? (y/n):",
44, 100);
		input = read_input();
		HAL_UART_Transmit(&huart3, (uint8_t *)nl, strlen(nl), 100);
```

```
		if (strcmp(input, "y") == 0) {
			res = f_open(&file, tokens[1], FA_CREATE_NEW);
		}
	} else if (res == FR_NO_PATH) {
		HAL_UART_Transmit(&huart3, (uint8_t *)"Could not find the path.", 24, 100);
		HAL_UART_Transmit(&huart3, (uint8_t *)nl, strlen(nl), 100);
	} else if (res == FR_OK) {
		while (f_gets(line, sizeof line, &file)) {
			HAL_UART_Transmit(&huart3, (uint8_t *)line, strlen(line), 100);
		}
		HAL_UART_Transmit(&huart3, (uint8_t *)nl, strlen(nl), 100);
	}
	f_close(&file);
}
```

Ukaz write

Za pisanje uporabimo funkcijo f_puts. Tu se je treba zavedati, da funkcija f_puts deluje tako, da v datoteko vstavi podatke tam, kjer je trenutno bralni/pisalni kazalec. Ko datoteko odpremo z f_open, je bralni/pisalni kazalec na začetku datoteke. Če takoj kličemo funkcijo f_puts, bodo novi podatki zapisani na začetek datoteke in tako povozili morebitne stare podatke.

```
FIL file;
char* input;
if (tokens[1] == NULL) {
	HAL_UART_Transmit(&huart3, (uint8_t *)"Please provide filename.", 24, 100);
	HAL_UART_Transmit(&huart3, (uint8_t *)nl, strlen(nl), 100);
} else {
	res = f_open(&file, tokens[1], FA_WRITE);
	if (res == FR_NO_FILE) {
		HAL_UART_Transmit(&huart3, (uint8_t *)"Could not find the file. Create file? (y/n):",
44, 100);
		input = read_input();
		HAL_UART_Transmit(&huart3, (uint8_t *)nl, strlen(nl), 100);
		if (strcmp(input, "y") == 0) {
			res = f_open(&file, tokens[1], (FA_CREATE_NEW | FA_WRITE));
			if (res != FR_OK) {
				return res;
			}
		} else {
			return res;
		}
	} else if (res == FR_NO_PATH) {
		HAL_UART_Transmit(&huart3, (uint8_t *)"Could not find the path.", 24, 100);
		HAL_UART_Transmit(&huart3, (uint8_t *)nl, strlen(nl), 100);
		return res;
	} else if (res != FR_OK) {
		return res;
	}
	HAL_UART_Transmit(&huart3, (uint8_t *)"Line to write to file: ", 24, 100);
	input = read_input();
	strcat(input, "\n\r");
	HAL_UART_Transmit(&huart3, (uint8_t *)nl, strlen(nl), 100);
	if (f_puts(input, &file) <= 0) {
		HAL_UART_Transmit(&huart3, (uint8_t *)err, strlen(err), 100);
```

```
		HAL_UART_Transmit(&huart3, (uint8_t *)nl, strlen(nl), 100);
	}
	f_close(&file);
}
```

Ukaz append

Za append prav tako uporabimo funkcijo f_puts. Razlika je v tem, da tukaj po odprtju datoteke uporabimo funkcijo f_lseek. To je funkcija, ki brez branja/pisanja prestavi bralni/pisalni kazalec. Kot drugi argument funkcije uporabimo kar funkcijo f_usize, ki nam vrne velikost datoteke. Tako postavimo bralni/pisalni kazalec takoj za že shranjenimi podatki. Zdaj lahko spet uporabimo f_puts in shranimo dodatni tekst.

```
FIL file;
char* input;
if (tokens[1] == NULL) {
	HAL_UART_Transmit(&huart3, (uint8_t *)"Please provide filename.", 24, 100);
	HAL_UART_Transmit(&huart3, (uint8_t *)nl, strlen(nl), 100);
} else {
	res = f_open(&file, tokens[1], FA_WRITE);
	if (res == FR_NO_FILE) {
		HAL_UART_Transmit(&huart3, (uint8_t *)"Could not find the file. Create file? (y/n):",
44, 100);
		input = read_input();
		HAL_UART_Transmit(&huart3, (uint8_t *)nl, strlen(nl), 100);
		if (strcmp(input, "y") == 0) {
			res = f_open(&file, tokens[1], (FA_CREATE_NEW | FA_WRITE));
			if (res != FR_OK) {
				return res;
			}
		} else {
			return res;
		}
	} else if (res == FR_NO_PATH) {
		HAL_UART_Transmit(&huart3, (uint8_t *)"Could not find the path.", 24, 100);
		HAL_UART_Transmit(&huart3, (uint8_t *)nl, strlen(nl), 100);
		return res;
	} else if (res != FR_OK) {
		return res;
	}
	res = f_lseek(&file, f_size(&file));
	if (res != FR_OK) {
		return res;
	}
	HAL_UART_Transmit(&huart3, (uint8_t *)"Line to write to file: ", 24, 100);
	input = read_input();
	strcat(input, "\n\r");
	HAL_UART_Transmit(&huart3, (uint8_t *)nl, strlen(nl), 100);
	if (f_puts(input, &file) <= 0) {
		HAL_UART_Transmit(&huart3, (uint8_t *)err, strlen(err), 100);
		HAL_UART_Transmit(&huart3, (uint8_t *)nl, strlen(nl), 100);
	}
	f_close(&file);
}
```

Ukaz mkdir

Za mkdir je na voljo funkcija f_mkdir, ki ustvari direktorij s podano potjo. Ime direktorija je lahko dolgo največ 8 znakov.

```
if (tokens[1] == NULL) {
	HAL_UART_Transmit(&huart3, (uint8_t *)"Please provide name.", 20, 100);
	HAL_UART_Transmit(&huart3, (uint8_t *)nl, strlen(nl), 100);
} else {
	res = f_mkdir(tokens[1]);
	if (res == FR_EXIST) {
		HAL_UART_Transmit(&huart3, (uint8_t *)"File/directory with this name already exists.",
45, 100);
		HAL_UART_Transmit(&huart3, (uint8_t *)nl, strlen(nl), 100);
	}
}
```

Ukaz cp

Za kopiranje uporabimo kombinacijo funkcij f_open, f_read, f_write. Potrebujemo buffer za posredovanje podatkov. Tu je pametno imeti še dve dodatni spremenljivki, ena za število bajtov prebranih, ena za število bajtov zapisanih.

V while zanki beremo in pišemo podatke in preverjamo, da se ti dve števili ujemata. Ko je prebranih bajtov enako nič, končamo.

```
FIL srcFile;
FIL dstFile;
BYTE buffer[4096];
UINT br, bw; // bytes read, bytes written
if (tokens[1] == NULL || tokens[2] == NULL) {
	HAL_UART_Transmit(&huart3, (uint8_t *)"Please provide source and destination filename.", 47,
100);
	HAL_UART_Transmit(&huart3, (uint8_t *)nl, strlen(nl), 100);
} else {
	res = f_open(&srcFile, tokens[1], FA_READ);
	if (res != FR_OK) {
		return res;
	}
	res = f_open(&dstFile, tokens[2], FA_WRITE | FA_CREATE_ALWAYS);
	if (res != FR_OK) {
		return res;
	}
	while(1) {
		res = f_read(&srcFile, buffer, sizeof buffer, &br);
		if (res != FR_OK || br == 0) {
			break;
		}
		res = f_write(&dstFile, buffer, br, &bw);
		if (res != FR_OK || bw < br) {
			break;
		}
	}
	f_close(&srcFile);
```

```
	f_close(&dstFile);
}
```

Ukaz mv

Tukaj lahko enostavno uporabimo funkcijo f_rename.

```
if (tokens[1] == NULL || tokens[2] == NULL) {
	HAL_UART_Transmit(&huart3, (uint8_t *)"Please provide current and new path/filename.", 45,
100);
	HAL_UART_Transmit(&huart3, (uint8_t *)nl, strlen(nl), 100);
} else {
	res = f_rename(tokens[1], tokens[2]);
	if (res == FR_EXIST) {
		HAL_UART_Transmit(&huart3, (uint8_t *)"File/directory with new path/name already
exists.", 49, 100);
		HAL_UART_Transmit(&huart3, (uint8_t *)nl, strlen(nl), 100);
	}
}
```

Ukaz rm

Tukaj lahko enostavno uporabimo funkcijo f_unlink. Pri tej funkciji je omejitev, da mora biti direktorij prazen.

```
if (tokens[1] == NULL) {
	HAL_UART_Transmit(&huart3, (uint8_t *)"Please provide file/directory name.", 35, 100);
	HAL_UART_Transmit(&huart3, (uint8_t *)nl, strlen(nl), 100);
} else {
	res = f_unlink(tokens[1]);
	if (res == FR_DENIED) {
		HAL_UART_Transmit(&huart3, (uint8_t *)"Directory must be empty.", 24, 100);
		HAL_UART_Transmit(&huart3, (uint8_t *)nl, strlen(nl), 100);
	}
}
```

Ukaza help in exit

Za konec dodamo še ukaz exit, ki samo prekine izvajanje ter ukaz help, ki izpiše ukaze in njihov kratek opis.

Zaključek

Tako opisan program nam enostavno omogoča delo z datotekami na USB napravi in predstavlja osnovo za naprednejše delo z USB napravo.

Pri samem uporabniškem vmesniku je pomembno, da vedno sporočimo, če pride do napake in da je opis napake čim bolj natančen. S tem pohitrimo reševanje težav v programu.

Demonstracija: https://video.arnes.si/watch/yfsb75v023dz

Hvala za pozornost.