Arduino ftp server i serijska komunikacija sa C# aplikacijom

Sadžaj

1 Zadatak projekta

Zadatak projekta bio je implementacija protokola uCFtp[1] za serversko-klijentsku komunikaciju. Serverska aplikacija izvršava se na mikrokontroleru *ESP32 Arduino* i napisana je u *Arduino IDE* programu pomoću *C* i *C++* programskog koda. Serverska aplikacija sadrži uCFtp interfejs koji koristimo na klijentskoj strani za manipulisanje podacima koji se nalaze na serverskoj strani. Podatke na serverskoj strani smeštamo na memorijsku karticu. Na klijentskoj strani imamo aplikaciju napisanu u programskom jeziku *C#*. Aplikacija je pisana u *Visual Studio Code*. Komunikacija između serverske i klijentske strane izvršava se bežično.

2 Mentor

Izrada projekta bila je u sklopu predmeta *Seminarski D* i mentor na ovom projektu bio je izabrani profesor Đorđe Herceg sa Prirodno-matematičkog fakulteta u Novom Sadu departman za Računarske nauke.

2.1 Reference

Većinu programskog koda koji koristimo na serverskoj strani predstavljaju kodovi koje je napisao profesor Đorđe Herceg.
Originalne kodove možemo naći na adresi: *https://github.com/djherceg*

[1] kvazi-FTP protokol za mikrokontrolere

3 Potrebni alati i povezivanje

3.1 Potrebni delovi

1. ESP32 Arudino mikrokontroler
2. Arudino modul za čitanje SD kartica
3. 6 žica sa golim krajevima
4. Opcionalno: lemilica

3.2 Programska okruženja

1. Arduino IDE
   Uputstvo za instalaciju: https://www.Arduino.cc/en/software
   Dodatne biblioteke i kompajlere koje je potrebno da instaliramo:
   - SdFat biblioteku verzija: 2.0.1.
     Biblioteku možemo instalirati direktno u programu *Arduino IDE* pomoću sledećih koraka:
     - Otvorimo `Tools->Manage Libraries`
     - Pretražujemo i instaliramo biblioteku sa nazivom SdFat autora Bill Greiman verzija 2.0.1.
   - WiFi biblioteku verzija: 1.2.7. Biblioteku možemo instalirati direktno u programu *Arduino IDE* pomoću sledećih koraka:
     - Otvorimo `Tools->Manage Libraries`
     - Pretražujemo i instaliramo biblioteku sa nazivom WiFi autora Arduino verzija 1.2.7.
   - ESP32 kompajler verzija: 1.0.3.
     Kompajler možemo instalirati na sledeći način. Podrazumevano *Arduino IDE* ne sadrži ponuđenu opciju za instalaciju ESP32 kompajlera. Potrebno je da dodamo repozitorijum:
     - Otvorimo `File>Preferences`
     - Dodamo u polje `Additional Board Manager URLs` sledeće linkove:
       https://dl.espressif.com/dl/package_esp32_index.json
       http://arduino.esp8266.com/stable/package_esp8266com_index.json

     Sada možemo da instaliramo kompajler:
     - Otvorimo `Tools->Board->Manage Boards`
     - Pretražujemo i instaliramo kompajler sa nazivom esp32 autora Espressif Systems verzija 1.0.3.
2. .NET Core
   Koristili smo .NET Core SDK 3.1.407
   Uputstvo za instalaciju: https://docs.microsoft.com/en-us/dotnet/core/install/
   Dodatne biblioteke koje je potrebno da instaliramo:
   - System.IO.Ports biblioteku verzija: 5.0.1
     Biblioteku možemo instalirati izvršavajući sledeću komandu:

     ```
     dotnet add package System.IO.Ports --version 5.0.1
     ```

3.3 Povezivanje mikrokontrolera i modula za čitanje SD kartica

Povezali smo pinove na sledeći način:

MicroSD modul	ESP32
3V3	3.3V
CS	GPIO 5
MOSI	GPIO 23
CLK	GPIO 18
MISO	GPIO 19
GND	GND

Povezivanje MicroSD modula sa ESP32 Arduino mikrokontrolerom

4 Opis klijentskog C# koda

4.1 Opis glavnog C# fajla: Program.cs

U fajlu *Program.cs* možemo videti primer upotrebe klijentskog koda pomoću kojeg možemo da šaljemo komande prema Arduinu i tumačimo rezultate izvršenih komandi.

Potrebno je da uvezemo u istom okruženju (*namespace*-u) sledeće *.cs* fajlove:

1. ArduinoCommands.cs (podsekcija 4.2.)
2. ArduinoInterpreter.cs (podsekcija 4.3.)
3. ArduinoIW.cs (podsekcija 4.4.)
4. ArduinoMessageBuilder.cs (podsekcija 4.5.)
5. CrcMath.cs (podsekcija 4.8.)
6. ErrorsInternal.cs (podsekcija 4.9.)
7. RequestTalk.cs (podsekcija 4.6.)
8. TalkBuffer.cs (podsekcija 4.7.)

U $Main$ metodi pokrećemo nit u kojoj čekamo, čitamo i obrađujemo dostupne podatke koji su primljeni sa komunikacionog tunela. Definišemo adresu Arduino servera sa kojim želimo da komuniciramo. Koristimo objekat klase $HttpRequestRW$ koji sadrži osnovne metode za komunikaciju sa serverom.

Nakon inicijalizacije objekta klase $HttpRequestRW$ pružamo klijentu mogućnost da unese komandu koju želi da pošaljemo serveru. Komandu šaljemo i čekamo na rezultat pomoću metode $SendCommandAndRecieveResult$ objekta klase $HttpRequestRW$. Nakon primljenog rezultata možemo da obradimo dostupne podatke. U glavnom programu pratimo da li objekat klase $CommandObject$ koji predstavlja trenutni rezultat izvršene komande zavisi od objekta koji predstavlja rezultat sledeće izvršene komande pomoću indikatora koji smo nazvali $waitingForTheChain$. Na primer komanda GET bez poslatog opcionalnog argumenta `crc` zavisi od sledeće izvršene komande (pogledati podsekciju 6.3.8). U ovom slučaju automatski šaljemo

sledeću komandu sa nazivom CRC i argumentima koji predstavljaju fajl-segment nad kojim je izvršena GET komanda.

Zavisnosti između dve uzastopne komande možemo da dodefinišemo (pogledati podsekciju *ArduinoCommands.cs*) .

Ukoliko imamo dostupne podatke za dalje obrađivanje, obrađujemo ih pomoću objekta koji predstavlja klasu *ArduinoIW.cs*. Za dodatna uputstva pogledati podsekciju 4.4. Obrađivanje vršimo pomoću metode $interpet()$. Ukoliko je obrađivanje bilo validano i rezultat ove metode predstavlja kompletan rezultat izvšene komande dalje proveravamo da li je rezultat zavistan od rezultata sledeće izvršene komande. U slučaju zavisnosti koristimo odgovarajuće provere da bi smo ispitali kompatibilnost između izvršenih komandi. Ukoliko nisu zavisni rezultati ispisujemo samo trenutni rezultat.

Ukoliko rezultat nije validan preskačemo ispisivanje. Takođe potrebno je da resetujemo graditelja koji sve vreme gradi rezultat izvršavanja komande i koji se nalazi u objektu koji predstavlja *ArduinoIW.cs* klasu.

Na kraju čuvamo rezultat trenutno izvršene komande u objektu koji predstavlja *ArduinoIW.cs* klasu.

4.2 ArduinoCommands.cs

Struktura fajla

```
(a)CommandObject
--->CommentCommandObject
--->DirCommandObject
--->ErrorCommandObject
--->MdCommandObject
--->CdCommandObject
--->PutCommandObject
--->GetCommandObject
--->DelCommandObject
--->CrcCommandObject
```

Opis

U fajlu $ArduinoCommands.cs$ definišemo validiranje pojedinačnih delova rezultata izvršene komande.

Opšti šablon rezultata izvršene komande po delovima može da izgleda:

1. header deo:
   - $\#OK\ arg1\ arg2$
   - ili $\#ERROR\ arg1\ arg2$
2. payload deo, sadrži linije koje počinju simbolom $>$. Primer:

   ```
   > file1 2
   > file2 10
   ```

3. footer deo, sadrži jednu reč, reč $\#END$ kojom označavamo da smo dobili celu poruku. Nakon reči $\#END$ u sledećem redu sledi simbol $ kojim označavamo da je Arduino spreman da prima nove komande.

Konkretan objekat koji dopunjuje i predstavlja implementaciju apstraktne-klase $CommandObject$ koristimo za predstavljanje rezultata izvršene komande.
U njemu deklarišemo i definišemo:

1. promenljive u kojima skladištimo prosleđene argumente za ponuđene parametre
2. da li je kompletiran
3. da li je validan
4. da li je zavistan od rezultata izvršavanja sledeće komande
5. metode u kojima obrađujemo linije payload-a
6. ostale pomoćne metode kao što su ispis rezultata i ostalo

Sada možemo detaljnije opisati i klase koje implementiraju ovu apstraktnu klasu. Za sve podržane komande na Arduinu imamo odgovarajuće klase koje koristimo za validaciju primljenog rezultata odgovarajuće izvršene komande. Implementirali smo sledeće klase:

1. $CommentCommandObject$
   obrađujemo komentar poslat sa Arduina.
2. $DirCommandObject$
   obrađujemo rezultat izvršavanja dir komande.
3. $ErrorCommandObject$
   obrađujemo rezultat izvršavanja er komande.
4. $MdCommandObject$
   obrađujemo rezultat izvršavanja md komande.
5. $CdCommandObject$
   obrađujemo rezultat izvršavanja cd komande.
6. $PutCommandObject$
   obrađujemo rezultat izvršavanja put komande.
7. $GetCommandObject$
   obrađujemo rezultat izvršavanja get komande.
8. $DelCommandObject$
   obrađujemo rezultat izvršavanja del komande.
9. $CrcCommandObject$
   obrađujemo rezultat izvršavanja crc komande.

U klasi $ArduinoCommands$ nabrojali smo nazive komandi.
Objekat klase $CommandObject$ sadrži i metode za definisanje zavisnosti između rezultata trenutno izvršene komande i sledećeg $CommandObject$ objekta koji će predstavljati rezultat sledeće izvršene komande. Koristimo sledeće metode za definisanje zavisnosti:

1. (virtuelnu) $checkChainCmdResult$ u kojoj možemo dodefinisati način provere zavisnosti

2. $getChainCommandName$ za vraćanje naziva komande od koje zavisi $this$ objekat

3. $checkChainCommandName$ za proveru podudarnosti naziva komandi

4. $isWaitingForTheChain$ u kojoj definišemo da li objekat koji predstavlja rezultat trenutno izvršene komande poseduje zavisnost sa objektom koji predstavlja rezultat sledeće izvršene komande

4.3 ArduinoInterpreter.cs

Struktura fajla

```
(a)ArduinoLineProccesor
--->HeaderLineProccessor
--->ResultLineProccessor
--->FooterLineProccessor
ProccessorObjectResult
ArduinoInterpreter
```

Opis

U fajlu $ArduinoIntepreter.cs$ definišemo klase i metode pomoću kojih obrađujemo poruku koju smo izgradili od podataka koji su se nalazili na baferu. Kao bazu imamo apstraktnu klasu $ArduinoLineProccesor$ u kojoj deklarišemo apstraktni metod $process$ u kojem konkretne klase navode proceduru obrađivanja svog dela poruke. Imamo klase-obrađivače za svaki deo poruke:

1. $HeaderLineProccessor$
   obrađujemo liniju poruke koja bi trebala da predstavlja header

2. $ResultLineProccessor$
   obrađujemo liniju poruke koja bi trebala da predstavlja liniju payload-a

3. $FooterLineProccessor$
   obrađujemo liniju poruke koja bi trebala da predstavlja footer.

Kao klasu vodilju imamo klasu $ArduinoInterpreter$ u kojoj obrađujemo celokupnu poruku koja je sastavljena od podataka sa bafera. Prvo obrađujemo header, nakon toga payload i na kraju footer i vraćamo odgovarajući rezultat.

Koristimo objekat klase $ProccessorObjectResult$ kao internu enkapsulaciju objekta klase $CommandObject$. Svrha klase $ProccessorObjectResult$ je kompatibilnije prenošenje rezultata između više obrađivača celokupne poruke.

4.4 ArduinoIW.cs

Struktura fajla

```
ArduinoIW
```

Opis

U ovom fajlu definišemo klase i metode za obrađivanje rezultata izvršene komande i slanje nove komande ka Arduinu. Na početku imamo objekat klase $ArduinoIW$ čijem konstruktoru prosleđujemo objekat klase $PortTalk$ u kojem definišemo tunel preko kojeg možemo da šaljemo podatke ka Arduinu. Korišćenjem metode $WriteLine$ nad objektom klase $PortTalk$ možemo da pošaljemo liniju teksta ka Arduinu i metodom $interpret$ možemo da obradimo rezultat izvršene komande. U metodi $interpret$ koristimo objekat klase $ArduinoInterpret$. Objekat klase $ArduinoInterpret$ sadrži odgovarajuće metode za obrađivanje pojedinačnih delova poruke. Najpre uzmemo sve podatke koje smo pročitali sa ($PortTalk$) tunela i sačuvali u ($TalkBuffer$) bafer i proveravamo da li trenutni podaci zadovoljavaju opšti šablon validne poruke. Celokupnu poruku gradimo pomoću objekta klase $ArduinoMessageBuilder$.

Ukoliko je zadovoljen opšti šablon validne poruke odnosno ukoliko pozivom metode $isComplete()$ nad objektom klase $ArduinoMessageBuilder$ dobijemo povratnu vrednost $true$ prosleđujemo celokupnu poruku na dalje obrađivanje. Sledeći korak obrade predstavlja korak građenja objekta koji predstavlja rezultat izvršene komande. Podržane komande možemo videti u sekciji $ArduinoCommands.cs$. Za ovaj korak koristimo metodu $proccess()$ objekta klase $ArduinoInterpreter$. Ukoliko uspešno obradimo podatke metodom $proccess()$ vratićemo objekat koji predstavlja rezultat izvršene komande. U suprotnom vraćamo vrednost $null$. U ovom slučaju povratna vrednost signalizira da nismo uspešno uspeli da validiramo podatke. Takođe u ovom slučaju resetujemo graditelja celokupne poruke pomoću $resetResultBuilder$.

Ukoliko objekat koji predstavlja rezultat izvršene komande nije validan ili je kompletiran potrebno je isto da resetujemo graditelja celokupne poruke.

U objektu klase $ArduinoInterpreter$ čuvamo objekat koji predstavlja rezultat prethodno izvršene komande. Koristimo $getPreviousCommandObject$ i $setPreviousCommandObject$ metode za postavljanje i vraćanje objekta koji predstavlja rezultat prethodno izvršene komande.

4.5 ArduinoMessageBuilder.cs

Struktura fajla

```
(a)MessagePartitionBuilder
--->HeaderPartitionBuilder
--->PayloadPartitionBuilder
--->FooterPartitionBuilder
--->ResultMessageBuilder
```

Opis

U ovom fajlu definišemo klase i metode za gradnju celokupne poruke koja bi trebala da predstavlja rezultat izvršene komande. Celokupna poruka sastoji se iz header-a, payload-a i footer-a. Imamo odgovarajuće graditelje i u svakom od njih različita stanja

u kojima mogu biti. U graditeljima ne ispitujemo validnost poruke već samo popunjavamo prazna mesta definisana šablonom opšte poruke. Imamo sledeće graditelje i stanja:

1. HeaderPartitionBuilder i stanja:
   (a) STATUS_MODE, stanje u kojem očekujemo prvu reč koja bi trebala da opisuje da li se komanda uspešno izvršila ili ne. Reči su razdvojeni razmakom.
   (b) CMD_NAME_MODE, stanje u kojem očekujemo drugu reč koja bi trebala da opisuje naziv komande koju smo pokušali da izvršimo.
   (c) FILE_PATH_MODE, stanje u kojem očekujemo treću reč koja bi trebala da opisuje fajl nad kojim smo pokušali da izvršimo komandu.
2. PayloadPartitionBuilder i stanja:
   (a) LINE_MODE, stanje u kojem očekujemo liniju payload-a
3. FooterPartitionBuilder i stanja:
   (a) STATUS_MODE, stanje u kojem očekujemo prvu reč koja bi trebala da se poklapa sa stringom $\#END$.

U svakom od graditelja gradimo sve dok ne naiđemo na liniju koja počinje simbolom koji predstavlja sledeću particiju poruke.

Kao glavnog dirigenta imamo objekat klase $ResultMessageBuilder$. Objekat klase $ResultMessageBuilder$ takođe sadrži metodu $build$ kojoj prosleđujemo po jednu liniju na obrađivanje. U metodi $build$ prosleđujemo dalje liniju odgovarajućem graditelju. Objekat klase $ResultMessageBuilder$ može biti u stanju:

1. HEADER_MODE, stanje predstavlja da trenutno gradimo header deo poruke
2. PAYLOAD_MODE, stanje predstavlja da trenutno gradimo payload deo poruke
3. FOOTER_MODE, stanje predstavlja da trenutno gradimo footer deo poruke
4. MSG_COMPLETE_MODE, stanje predstavlja da smo uspešno izgradili celu poruku i naišli na simbol $. Simbol $ predstavlja indikator da je Arduino spreman da prima nove komande.

4.6 RequestTalk.cs

Struktura fajla

```
(a)HttpRequestRW
--->PostHttpRequestRW
```

Opis

U ovom fajlu definišemo klasu $HttpRequestRW$ u kojoj deklarišemo osnovne metode za komunikaciju preko tunela. Klase koje nasleđuju $HttpRequestRW$ definišu način komunikacije preko tunela.

Metode u klasi $HttpRequestRW$ smo definisali na sledeći način:

1. U metodi $setup()$ inicijalizujemo adresu Arduino servera sa kojim želimo da komuniciramo.

2. Deklarisali smo i definisali praznu virtuelnu asihronu metodu `SendCommandAndRecieveResult`. Preporučljivo je da u klasama u kojima nasleđujemo klasu $HttpReqeustRW$ definišemo logiku za slanje konkretnog `HTTP` zahteva.

Definisali smo konkretnu klasu $PostHttpRequestRW$. U klasi $PostHttpRequestRW$ nasleđujemo i definišemo sve osnovne metode iz baze klase $HttpRequestRW$. Pomoću metoda $SendCommandAndRecieveResult(string)$ objekta klase $PostHttpRequestRW$ možemo da šaljemo i čitamo rezultate `POST` zahteva. Koristimo $HttpClient$ iz paketa $System.Net.Http$ unutar metoda $SendCommandAndRecieveResult(string)$.

4.7 TalkBuffer.cs

Struktura fajla

`TalkBuffer`

Opis

U ovom fajlu definišemo klasu i metode pomoću kojih čuvamo pročitane podatke sa tunela. Objekat klase $TalkBuffer$ sadrži sledeće metode i promenljive za manipulisanje baferom:

1. promenljivu $position$ koristimo da označimo broj pročitanih podataka iz bafera
2. promenljivu $length$ koristimo da označimo dužinu bafera
3. promenljivu $maxbuf$ koristimo da označimo maksimalnu dužinu bafera
4. metodu $read()$ koristimo za čitanje bafera na $position$ poziciji
5. metodu $isnext()$ koristimo da proverimo da li smo naišli na kraj bafera
6. metodu $clear()$ koristimo za vraćanje na početno stanje
7. metodu $init(string)$ koristimo za inicijalizaciju bufera
8. metodu $isoverflow()$ za proveru prekoračenja bafera

4.8 CrcMath.cs

Struktura fajla

`CrcMath`

Opis

U ovom fajlu definišemo metode za izračunavanje CRC16 vrednosti za prosleđen string ili celobrojni broj. Algoritami koji koristimo za izračunavanje CRC16 vrednosti se poklapa sa algoritmom koji je trenutno u upotrebi na Arduino platformi.

4.9 ErrorsInternal.cs

Struktura fajla

`ErrorsApp`

Opis

U ovom fajlu definišemo strukture za čuvanje grešaka. Greške mogu da se jave:

1. interno odnosno u procesu validiranja poruke koja predstavlja rezultat izvršene komande poslate sa serverske strane
2. eksterno odnosno greške koje su nastale na serverskoj strani u procesu obrađivanja poslate komande

Koristimo globalno statičnu promenljivu $errno$ u kojoj smeštamo celobrojnu vrednost trenutne greške. U bilo kojem delu koda možemo da pozovemo statičnu metodu $ErrorsApp.get()$ kojom vraćamo vrednost tipa $string$ kojom opisujemo trenutnu grešku. Podržane greške čuvamo u niz sa nazivom $errors$. Rezervisali smo:

1. od [0-9] mesta za interne greške. Interne greške čuvamo u $enum$ sa nazivom $ErrnoInternalCodes$. Prva greška ima vrednost 0 i svaka naredna za po jednu veću. Lista sadrži:
   (a) $STATUS\_PROCCESSOR\_ERROR\_NOT\_VALID\_HEADER$ - greškom označavamo da header poruke nije validan odnosno ne ispunjava šablon validnog header-a
   (b) $STATUS\_PROCCESSOR\_ERROR\_NOT\_VALID\_PAYLOAD$ - greškom označavamo da payload poruke nije validan odnosno ne ispunjava šablon validnog payload-a
   (c) $STATUS\_PROCCESSOR\_ERROR\_NOT\_VALID\_FOOTER$ - greškom označavamo da footer poruke nije validan odnosno ne ispunjava šablon validnog footer-a
   (d) $STATUS\_PROCCESSOR\_ERROR\_UNKNOWN\_CMD$ - greškom označavamo da header poruke za drugi parametar (naziv komande koja je izvršena) sadrži argument naziv komande koja nije podržana
   (e) $RESULT\_PROCCESSOR\_ERROR\_COMMAND\_PARSE$ - greškom označavamo da payload poruke nije validan odnosno ne ispunjava šablon validnog payload-a
   (f) $RESULT\_PROCCESSOR\_ERROR\_RESULT\_LINE\_MISSING$ - greškom opisujemo konkretnije da linija koja predstavlja deo payload poruke ne počinje simbolom $>$.
2. od [10-99] mesta za eksterne greške. Eksterne greške čuvamo u $enum$ sa nazivom $ErrnoExternalCodes$. Prva greška ima vrednost 10 i svaka naredna za po jednu veću. Lista sadrži:
   (a) od [10-49] predstavljaju mesta za greške koje su nastale prilikom obrade poslate komande na serverskoj strani:
      i. $CMD\_ERR\_GENERALERROR$ - predstavlja opštu grešku

ii. $CMD\_ERR\_MISSINGARGUMENT$ -
greška ukazuje na nedostatak argumenta za odgovarajući parametar
iii. $CMD\_ERR\_EXTRAARGUMENT$ -
greška ukazuje na prekoračenje broja prosleđenih argumenta
iv. $CMD\_ERR\_INVALIDVALUE$ -
greška ukazuje da prosleđeni argument nije validan
v. $CMD\_ERR\_VALUEOUTOFRANGE$ -
greška ukazuje da prosleđeni argument se ne nalazi u skupu dozvoljenih vrednosti
vi. $CMD\_ERR\_UNKNOWNCOMMAND$ -
greška ukazuje da prosleđeni argument koji odgovara parametru naziva komande koju želimo da izvršimo predstavlja naziv komande koja nije podržana na Arduinu

(b) ostalih [50-99] grešaka predstavljaju greške koje su nastale prilikom manipulisanja podacima na Arduinu prilikom obrade poslate komande.

5 Pokretanje programa i uključivanje projektnog koda u ostalim projektima

5.1 Uputstvo za pokretanje programa

1. Otvaramo .ino fajl u Arduino IDE koji se nalazi u folderu ArduinoFiles/Program
2. Presnimimo program na Arduino mikrokontroler
3. Pokrećemo program kojim možemo da čitamo i interpretiramo rezultate izvršene komande. Program se nalazi u folderu NETCore/Program/ReadProgram i možemo ga pokrenuti pomoću dotnet komande:

   ```
   dotnet run
   ```

4. Pokrećemo program kojim možemo da šaljemo komande ka Arduinu. Program se nalazi u folderu NETCore/Program/WriteProgram.

Sada možemo da šaljemo komande ka Arduinu i čitamo rezultate izvršenih komandi.

5.2 Uputstvo za korišćenje projektnog koda u ostalim projektima

5.2.1 Arduino projektni kod

Potrebno je da obezbedimo i uvezemo odgovarajuće fajlove koji su opisani u sekciji 7.1 sa nazivom *Opis glavnog Arduino koda - SerijskiKomandniProcesorSDCARD.ino*.

U glavnom programu uvozimo sledeće fajlove:

- utilis.h
- cmdProc.h
- sdCardManipulator.h

- returnValues.h
- ByteArray.h
- recieveCommand.h

U sekciji 7.1 možemo videti logiku iza upotrebe klasa, metoda i promenljivih koji su sadržani u gore navedim fajlovima.

5.2.2 .NET projektni kod

Potrebno je da obezbedimo i uvezemo u radno okruženje (*namespace*) sve fajlove koji su opisani u sekciji 4.1 sa nazivom *Opis glavnog C# koda - Program.cs*.

Takođe potrebno da uradimo i sledeće:

1. dodamo u fajlu sa nazivom *.csproj* sledeće linije:

```
<ItemGroup>
<ProjectReference Include=
"..\\NETCore\Program\ReadProgram\coreapp.csproj" />
</ItemGroup>
```

2. uvezemo paket $System.IO.Ports$:

```
<ItemGroup>
<PackageReference Include="System.IO.Ports" Version="5.0.1" />
</ItemGroup>
```

6 Uputstvo za korišćenje komandi

6.1 Značenje simbola

$	Nakon izvršavanja tekuće naredbe signaliziramo da smo spremni za prijem nove naredbe i šaljemo simbol $ klijentskoj strani
#END	Signaliziramo korisniku da je prethodni ispis završen
@	Poruke informativnog karaktera sadrže kao prvi znak "@"
>	Linije payload-a imaju prvi znak ">"
#	Linije koje počinju ovim simbolom označavaju status

Table 1: Značenje simbola

Prilikom slanja komande ka Arduinu pored naziva komande navodimo i argumente za odgovarajuće parametre koji su podržani za navedenu komadnu.

6.2 Povratne vrednosti u opštem slučaju

- Slučaj greške

```
#ERROR <command-name> ERROR_CODE
>
#END
$
```

$ERROR\_CODE$ predstavlja celobrojni broj kojim opisujemo grešku nastalu pri izvršavanju komande sa nazivom `<command-name>`.

- Uspešno izvršavanje

```
#OK <command-name> <file-path>
> <function-results-list>
#END
$
```

Značenje parametara:

1. `command-name` predstavlja naziv komande koju smo uspešno izvršili
2. `file-path` predstavlja putanju fajla ili foldera nad kojim smo uspešno izvršili

komandu.

6.3 Lista podržanih komandi i povratne vrednosti

6.3.1 HELP

Koristimo za vraćanje uputstva za korišćenje podržanih komandi.
Upotreba

```
help
```

Povratne vrednosti

- Uspešno izvršavanje

```
#OK help <cwd>
> <available-commands-list>
#END
$
```

1. `cwd` predstavlja putanju trenutno radnog direktorijuma
2. `available-commands-list` predstavlja listu podržanih komandi

Primer:

```
[\folder1\folder2]: help
#OK help \folder1\folder2
> <available-commands-list>
#END
$
```

6.3.2 DIR

Koristimo za vraćanje liste fajlova sa dužinama iz tekućeg foldera. Prilikom uspešnog izvršavanja vraćamo $\#OK$ i ime tekućeg foldera i zatim u narednim redovima listu fajlova sa dužinama. Upotreba

```
dir
```

Povratne vrednosti

- Slučaj greške

  ```
  #ERROR dir ERROR_CODE
  >
  #END
  $
  ```

  Lista kodova grešaka:

50	SdFat open: Unable to open the current working directory

  Table 2: Značenje kodova grešaka

- Uspešno izvršavanje

  ```
  #OK dir <folder-path>
  ><folder-listing>
  #END
  $
  ```

  1. `folder-path` predstavlja naziv direktorijuma koji smo uspešno pročitali
  2. `folder-listing` predstavlja listu fajlova koji se nalaze u pročitanom folderu

Primer:

```
[\folder1\folder2]: dir
#OK dir \folder1\folder2
>10 fajl1
>13 fajl2
>0 folder3\
#END
$
```

6.3.3 CD

Vraćamo tekući folder sa poslatom CD komandom bez argumenta ili zadajemo tekući folder ukoliko je poslata komanda sa odgovarajućim argumentom. Ukoliko je argument "/" postavljamo tekući folder na root.
Upotreba:

```
cd [<relative-path>]
```

Povratne vrednosti

- Slučaj greške

```
#ERROR cd ERROR_CODE
>
#END
$
```

52	SdFat chdir: Can't change to the folder (Check if the folder exists)

Table 3: Značenje kodova grešaka

- Uspešno izvršavanje

```
#OK cd <cwd-path>
>
#END
$
```

1. `cwd-path` predstavlja putanju trenutno radnog direktorijuma

Primer:

```
[\folder1\folder2]: cd folder3
#OK \cd folder1\folder2\folder3
>
#END
$
```

6.3.4 MD

Kreiramo novi folder unutar tekućeg foldera. Vraćamo $\#OK$ i ime foldera ako prilikom uspešnog izvravanja ili $\#ERROR$ ukoliko se javila greška.
Upotreba

```
md <folder-name>
```

Povratne vrednosti

- Slučaj greške

```
#ERROR md ERROR_CODE
>
#END
$
```

53	SdFat exists: File with the same name already exists.
55	SdFat mkdir: Can't make a folder

Table 4: Značenje kodova grešaka

- Uspešno izvršavanje

```
#OK md <folder-path>
>
#END
$
```

1. `folder-path` predstavlja putanju novog uspešno napravljenog direktorijuma

Primer:

```
[\folder1\folder2]: md folder4
#OK md \folder1\folder2\folder4
>
#END
$
```

6.3.5 DEL

Brišemo zadati folder ili fajl unutar tekućeg foldera. Ako ne možemo da obrišemo vraćamo grešku. Vraćamo $\#OK$ i ime fajla prilikom uspešnog izvršavanja ili $\#ERROR$ ako se javila greška.
Upotreba

```
del <file-name>
```

Povratne vrednosti

- Slučaj greške

```
#ERROR del ERROR_CODE
>
#END
$
```

54	SdFat exists: File doesn't exists
50	SdFat open: Can't access to the file
56	SdFat rmRfStar: Can't delete the folder
57	SdFat remove: Can't delete the file

Table 5: Značenje kodova grešaka

- Uspešno izvršavanje

```
#OK del <file-name>
>
#END
$
```

1. `file-name` predstavlja putanju fajla koji smo uspešno izbrisali

Primer:

```
[\folder1\folder2]: del folder4
#OK del \folder1\folder2\folder4
>
#END
$
```

6.3.6 CRC

CRC komandom izračunavamo CRC16 checksum fajl segmenta i opcionalno ga upoređujemo sa prosleđenim odgovarajućim argumentom. Vraćamo dužinu fajla i CRC16 vrednost fajl segmenta kao dva celobrojna broja razdvojena razmakom.

1. Upotreba 1

   ```
   crc <file-name> <start> <length>
   ```

2. Upotreba 2

   ```
   crc <file-name> <crc-value>
   ```

3. Upotreba 3

   ```
   crc <file-name> <start> <length> crc-value
   ```

1. `file-name` - predstavlja ime fajla za čiji segment želimo da izračunamo crc
2. `start` - predstavlja startnu poziciju sa koje želimo da počnemo računanje CRC vrednosti
3. `length` - predstavlja broj karaktera nakon startne pozicije nad kojima želimo da izračunamo CRC vrednost
4. `crc-value` - predstavlja CRC vrednost koju želimo da proverimo da li je jednaka izračunatoj CRC vrednosti na serverskoj strani za odgovarajući segment fajla sa nazivom `file-name`. Ukoliko ne navedemo argument za parametre `start` i `length` podrazumevano računamo CRC vrednost za ceo fajl.

Ukoliko ne navedemo argument za parametre `start` i `length` podrazumevano uzimamo za $start = 0$ i za $length$ dužinu celog `file-name` fajla.

Povratne vrednosti

- Slučaj greške

```
#ERROR crc ERROR_CODE
>
#END
$
```

54	SdFat exists: File doesn't exists
50	SdFat open: Can't access to the file
56	SdFat rmRfStar: Can't delete the folder
57	SdFat remove: Can't delete the file

Table 6: Značenje kodova grešaka

- Uspešno izvršavanje - ishod 1

```
#OK crc <file-path>
><start> <length>
><file-length> <crc-value>
#END
$
```

1. `start` predstavlja startnu poziciju sa koje smo počeli računanje CRC vrednosti
2. `length` predstavlja broj karaktera nakon startne pozicije nad kojima je smo izračunali CRC vrednost
3. `file-path` predstavlja putanju fajla nad kojim smo za segment $[start - (start + length - 1)]$ uspešno izračunali CRC vrednost
4. `crc-value` predstavlja CRC vrednost za segment $[start-(start+length-1)]$

- Uspešno izvršavanje - ishod 2

```
#OK crc <file-path>
><start> <length>
><file-length> <0>
#END
$
```

1. Ovaj ishod se razlikuje od ishoda 1 samo u poslednjem podatku `<0>` kojim označavamo da CRC vrednost koju smo prosledili sa klijentske strane nije jednaka izračunatoj CRC vrednosti na serverskoj strani

- Uspešno izvršavanje - ishod 3

```
#OK crc <file-path>
><start> <length>
><file-length> <1>
#END
$
```

1. Ovaj ishod se razlikuje od ishoda 1 samo u poslednjem podatku <1> koji označavamo da CRC vrednost koju smo prosledili sa klijentske strane jednaka je izračunatoj CRC vrednosti na serverskoj strani

Primer 1:

```
[\folder1\folder2]: crc fajl1
#OK crc \folder1\folder2\fajl1
>687 19238
#END
$
```

Primer 2:

```
[\folder1\folder2]: crc fajl1 19238
#OK crc \folder1\folder2\fajl1
>687 1
#END
$
```

6.3.7 PUT

Snimamo n bajtova u zadati fajl. Bajtovi su zadati kao celobrojne vrednosti u ASCII kodu. Primer PUT "fajl.txt" 65 66 67 69 –> u fajlu će se naći slova "ABCE". Ako fajl ne postoji, kreiramo ga. Ako postoji bajtove dodajemo na njega.
Upotreba

```
put <file-name> <array-of-dec-values>
```

Povratne vrednosti

- Slučaj greške

```
#ERROR put ERROR_CODE
>
#END
$
```

- Uspešno izvršavanje

```
#OK put <file-name>
><written-data-length>
#END
$
```

50	SdFat open: File exists but we couldn't open the file (check if the folder with same name exists.)
51	SdFat open: Can't create file.
61	Parsing error: Not valid argument. (Unsuccessful decimal value parsing)

Table 7: Značenje kodova grešaka

1. `written-data-length` predstavlja broj uspešno upisanih podataka
2. `file-name` predstavlja naziv fajla u kojem smo uspešno upisali `written-data-length` podataka

Primer:

```
[\folder1\folder2]: put new_file 65 66
#OK put \folder1\folder2\new_file
>2
#END
$
```

6.3.8 GET

Vraćamo sadržaj fajl segmenta u vidu niza celobrojnih brojeva. Počinjemo čitanje od pozicije $start$ i čitamo $length$ bajtova. Na kraju niza bajtova opcionalno dodajemo CRC16 vrednost za pročitani blok.

1. Upotreba 1

   ```
   get <file-name>
   ```

2. Upotreba 2

   ```
   get <file-name> <start> <length>
   ```

3. Upotreba 3

   ```
   get <file-name> "crc"
   ```

4. Upotreba 4

   ```
   get <file-name> <start> <length> "crc"
   ```

1. `file-name` - predstavlja ime fajla koji želimo da čitamo
2. `start` - predstavlja startnu poziciju sa koje želimo da počnemo čitanje
3. `length` - predstavlja broj karaktera nakon startne pozicije koje želimo da pročitamo
4. `"crc"` - predstavlja doslovan argument. Ovim argumentom označavamo da želimo da izračunamo CRC vrednost za odgovarajući segment fajla sa nazivom `file-name`. Ukoliko ne navedemo argument za parametre `start` i `length` podrazumevano računamo CRC vrednost za ceo fajl.

Upotreba

```
get <file-name> [<start> <length>] ["crc"]
```

Povratne vrednosti

- Slučaj greške

```
#ERROR get ERROR_CODE
>
#END
$
```

66	Not valid argument. (Unsuccessful decimal value parsing for the <start>argument)
67	Not valid argument. (Unsuccessful decimal value parsing for the <length>argument)
68	Not valid arguments. (<start>+ <length>is greater then the size of the file)
69	Not valid arguments. (The <start>argument is not null but the <length>argument is null)
52	SdFat open: File exists but we couldn't open the file (check if the folder with same name exists.)
60	Fgets: Can't read from a file
61	Fgets: Can't read from a file [line too long]
54	SdFat exists: File doesn't exist

Table 8: Značenje kodova grešaka

- Uspešno izvršavanje - ishod 1

```
#OK get <file-name>
><start> <length>
><file-sector-data>
#END
$
```

1. `start` predstavlja startnu poziciju sa koje smo počeli čitanje
2. `length` predstavlja broj karaktera nakon startne pozicije koje smo pročitali
3. `file-name` predstavlja putanju fajla za koji smo pročitali segment $[start-(start+length-1)]$
4. `file-sector-data` predstavlja podatke iz segmenta $[start-(start+length-1)]$

- Uspešno izvršavanje - ishod 2

```
#OK get <file-name>
><start> <length>
><file-sector-data>
><crc-value>
#END
$
```

1. Ovaj ishod se razlikuje od ishoda 1 samo u poslednjem podatku `<crc-value>` koji predstavlja CRC vrednost za segment $[start - (start + length - 1)]$

Primer:

```
[\folder1\folder2]: get new_file 0 2 crc
#OK get \folder1\folder2\new_file
>65 66
>13123
#END
$
```

6.4 Kodovi ostalih grešaka

COMMAND_ERR_GENERALERROR	10
COMMAND_ERR_MISSINGARGUMENT	11
COMMAND_ERR_EXTRAARGUMENT	12
COMMAND_ERR_INVALIDVALUE	13
COMMAND_ERR_VALUEOUTOFRANGE	14
COMMAND_ERR_UNKNOWNCOMMAND	15

Table 9: Ostale greške

Značenje ovih kodova možemo pogledati u podsekciji 4.9.

7 Opis serverskog Arduino koda

7.1 Opis glavnog Arduino fajla: SerijskiKomandniProcesorSDCARD.ino

Potrebno je da uvezeno instaliranu biblioteku *SdFat* (pogledati sekciju refl1).

Takođe uvozimo i sledeće fajlove:

- utilis.h (podsekcija 7.3.)
- cmdProc.h (podsekcija 7.5.)
- sdCardManipulator.h (podsekcija 7.8.)
- returnValues.h (podsekcija 7.7.)
- ByteArray.h (podsekcija 7.4.)
- recieveCommand.h (podsekcija 7.2.)

U nastavku se nalaze odgovarajuće podsekcije u kojima opisujemo način upotrebe ovih fajlova.

Koristimo tri interne promenljive:

- Objekat klase $CmdProc$. Klasu deklarišemo u fajlu sa nazivom "cmdProc.h". U objektu klase $CmdProc$ čuvamo podržane komande
- Objekat klase Buf koristimo za čuvanje pročitanih podataka sa tunela
- Promenljivu $chipSelect$ u kojoj čuvamo celobrojnu vrednost pina preko kojeg vršimo vremensku sinhronizaciju sa modulom za čitanje SD kartica.

Takođe koristimo i globalne promenljive koje smo uvezli iz fajla "sdCardManipulator.h":

- Objekat sa nazivom sd klase $SdFat32$. Objekat klase $SdFat32$ predstavlja "kernel" za upravljanje modulom za čitanje SD kartica
- Objekat sa nazivom $cwdFile$ klase $File32$. Objekat klase $File32$ predstavlja fajl koji je trenutno otvoren odnosno sa kojim trenutno radimo i koji se nalazi na SD kartici.

Dopunjujemo metode:

1. $setup()$. Postavljamo brzinu protoka serijskog čitanja i pisanja na 9600 baud i nakon toga pokušavamo da inicijalizujemo *WiFi* server. U slučaju uspešne postavke servera inicijalizujemo i tunel za manipulisanje podacima između Arduino mikrokontrolera i modula za čitanje SD kartica. U slučaju neuspešne inicijalizacije prijavljujemo grešku. Takođe ukoliko nismo uspeli da otvorimo *root* korišćenjem metode $cwdFile.open("/")$ nad objektom klase $File32$ prijavljujemo grešku. Nakon završetka rada zatvaramo fajl korišćenjem metode $cwdFile.close()$.

   Ostalo je još da inicijalizujemo i ponudimo programeru podržane komande. Koristimo objekat klase $CmdProc$ u kome čuvamo podržane komande. Na početku izvršavamo metodu $Init(int)$ kojoj prosleđujemo za argument broj podržanih komandi. Nakon toga dodajemo komande korišćenjem metode $Add(string, (*cmdCallback), int, int)$ gde

   - za prvi argument navodimo ime komande
   - za drugi argument pokazivač na funkciju koja treba da se izvrši
   - broj obaveznih argumenta
   - broj ukupnih argumenata

2. $loop()$ u kojoj navodimo logiku za čitanje i interpretiranje pročitanog tunela. Koristimo objekat klase $RecieveAndAnswerCommand$ i metod $proccess()$. Unutar svakog sistemskog poziva metode $loop()$ pozivamo metodu $process()$.

   U metodi $proccess()$ čekamo klijenta da se poveže. Ukoliko imamo dostupnog klijenta čitamo zahtev. Definisali smo uslov za zahtev. Zahtevu je potrebno da sadrži sledeće parametre (navedenim redosledom ne nužno sekvencijalnim):

   - `POST / HTTP/1.1 200 OK`
   - `Content-Type: text/plain`

- `Content-Length: <payload-length>`
  `<new-line>`
  `<payload>`

Ostale linije su ignorisane. Parametar `<paylod-length>` označava dužinu `payload`-a u bajtovima. Parametar `payload` predstavlja poslatu komandu za obrađivanje. Primljenu i pročitanu komandu smeštamo u prethodno definisanu promenljivu $sbuf$. Bafer $sbuf$ prosleđujemo na obradu. Obrađivanje vršimo korišćenjem statične metode $Parse(Buf)$ klase $Commands$. Zavisno od ishoda obrađivanja zahteva pošiljaocu vraćamo $HTTP$ odgovor sa celobrojnim statusom:

- 404 (neuspešan ishod) praćen sa `payload`-om u kome je sadržan opis greške
- 200 (uspešan ishod) praćen sa `payload`-om u kome je sadržan rezultat obrađene komande

7.2 recieveCommand.h

U ovom fajlu navodimo klasu sa nazivom $RecieveAndAnswerCommand$ u kojoj definišemo metode i promenljive koje za slanje $HTTP$ zahteva ka serveru. Konstruktoru objekta klase $RecieveAndAnswerCommand$ prosleđujemo:

- instancu servera klase $WiFiServer$
- listu podržanih komandi klase $CmdProc$
- indikatore postojanja grešaka koje su vezane za inicijalizaciju komunikacije sa modulom za čitanje memorijske kartice.

U klasi $RecieveAndAnswerCommand$ definisali smo metodu $interpret$ u kojoj obrađujemo poslatu komandu sa klijentske strane.

Poslatu komandu obrađujemo i rezultat šaljemo kao $HTTP$ odgovor. $HTTP$ odgovor sadrži status kod 200 i tip sadržaja $text/html$. Rezultat obrađivanja poslate komande okružujemo $html$ oznakama i smeštamo u $payload$ $HTTP$ odgovora.

7.3 utilis.h

U ovom fajlu navodimo klasu sa nazivom $ByteArray$ u kojoj definišemo pomoćne metode za parsiranje i ispitivanje raznih podataka:

1. $tryParseInt(char*, int\&)$ koristimo za konvertovanje prvog prosleđenog argumenta koji predstavlja reč u celobrojnu ili heksadecimalnu vrednost. Rezultat smeštamo na memorijsku lokaciju na kojoj se nalazi drugi argument.

2. $tryParseDec(char*, int\&, bool)$ koristimo za konvertovanje prvog prosleđenog argumenta koji predstavlja reč u decimalnu vrednost. Rezultat smeštamo na memorijsku lokaciju na kojoj se nalazi drugi argument. Kao treći opcionalni argument imamo indikator da li prvi argument na prvom mestu sadrži karakter $'\_'$.

7.4 ByteArray.h

U ovom fajlu navodimo klasu sa nazivom $ByteArray$ u kojoj definišemo metode i promenljive koje koristimo za čuvanje pročitanih podataka sa tunela.

7.5 cmdProc.h

U ovom fajlu deklarišemo klase koje koristimo za pravljenje objekata u kojima čuvamo i koristimo podržane komande.

U klasi Cmd definišemo okruženje za izvršavanje komandi. U objektu klase Cmd čuvamo naziv komande, memorijsku lokaciju funkcije koju je potrebno da izvršimo, broj obaveznih parametara i ukupan broj parametara.

U objektu klase $CmdProc$ čuvamo sve podržane komande. Komande dodajemo korišćenjem metode $Add(char, (*cmdCallback), int, int)$. Objekat klase $CmdProc$ sadrži metode $GetNextToken()$, $Parse(char*)$ i $Exec()$ koje koristimo za određivanje konteksta u kojem izvršavamo funkciju komande odnosno određujemo argumente koje je potrebno da prosledimo funkciji komande. Ukoliko se broj argumenta poklapa sa traženim brojem argumenata izvršavamo funkciju komande.

7.6 cwdPath.h

U objektu klase $SdFat32$ biblioteke $SdFat$ ne postoji praćenje apsolutne putanje trenutnog radnog direktorijuma. Definisali smo novu klasu $CwdPath$. Objekat klase $CwdPath$ koristimo za praćenje apsolutne putanje trenutnog radnog direktorujima.

7.7 returnValues.h

U ovom fajlu definišemo metode u kojima modeliramo povratne vrednosti. Model rezultata izvršavanja primljene komande prati šablone povratnih vrednosti u opštem slučaju koje smo definisali u podsekciji 6.2 - *Povratne vrednosti u opštem slučaju*.

7.8 sdCardManipulator.h

U ovom fajlu definišemo funkcije komandi. Najpre uvozimo potrebne fajlove:

1. <arduino.h> i <avr/pgmspace.h> uvozimo za čitanje podataka iz programske memorije
2. <string.h> za manipulaciju stringovima
3. <SdFat.h> za manipulaciju podataka preko modul čitača za SD kartice
4. <util/crc16.h> za računanje CRC16 checksum-e
5. "cwdPath.h" za praćenje apsolutne putanje trenutno radnog direktorijuma
6. "returnValues.h" u ovom fajlu se nalaze modeli povratnih vrednosti

U programsku memoriju smeštamo

- sadržaj koji koristi $HELP$ komanda
- nazive podržanih komandi

U nastavku su opisane funkcije podržanih komandi. Sve funkcije imaju parametar koji predstavlja pokazivač na objekat klase $CmdProc$. U ovom objektu nalazi se kontekst u kojem se funkcija izvršava.

Na početku svake funkcije kopiramo naziv komande iz programske memorije u izvršni deo koda korišćenjem metode $strcpy\_P$. Nakon izvršavanja komande zatvaramo $cwdFile$ fajl nad kojim smo pokušali da izvšrimo komandu i štampamo zaglavlje i simbol $ korišćenjem statične metode $printEndAndReady$ klase $ReturnValues$.

7.8.1 cmdDir

Definisali smo dve promenljive sa nazivima i_file i $cwdFile$ klase $File32$. Korišćenjem metode $open(char*)$ objekta $cwdFile$ otvaramo trenutni radni direktorijum. U slučaju greške prijavljujemo i vraćamo grešku.

Metodom $rewind()$ premotovamo $cwdFile$ na početak i $while$ petljom iteriramo kroz trenutni radni direktorijum. Iterativno postavljamo i_file objekat da predstavlja fajl koji sledeći čitamo. Postavljamo ga korišćenjem metode $openNext$ kojoj prosleđujemo direktorijum koji čitamo i tip privilegije čitanja.

Prilikom svake iteracije:

1. ispisujemo simbol > korišćenjem statične metode $printResult$ klase $ReturnValues$
2. ispisujemo veličinu fajla
3. ispisujemo ime fajla i u slučaju direktorijuma dodajemo na kraju simbol "/"
4. na kraju svake iteracije zatvaramo i_file.

7.8.2 cmdHelp

Koristimo klasu $\_\_FlashStringHelper$ pomoću koje čitamo reč iz programske memorije. Ovo je alternativni način i ovo možemo da uradimo i korišćenjem metode $strcpy\_P$.

Iteriramo kroz niz sa nazivom $helpTable$ koji je smešten u programsku memoriju i u svakoj iteraciji čitamo element niza iz programske memorije i štampamo ga.

7.8.3 cmdPut

Obrađujemo prosleđene argumente. Proveravmo da li fajl sa nazivom prvog prosleđenog argumenta postoji. Ako postoji pokušavamo da ga otvorimo i u suprotnom pokušavamo da ga napravimo. Prilikom neuspešnog izvršavanja vraćamo odgovarajuću grešku.

Drugi i ostali argumenti predstavljaju podatke koje želimo da upišemo u fajl. Svaki argument obrađujemo i proveravamo da li predstavlja celobrojnu vrednost. U slučaju celobrojne vrednosti upisujemo argument u fajl.

7.8.4 cmdDel

Ispitujemo da li fajl postoji i ukoliko postoji pokušavamo da ga obrišemo. Ukoliko nam je prosleđen naziv foldera koji trebamo da obrišemo koristimo metodu $rmRfStar$ objekta klase $File32$ za rekurzivno brisanje ne nužno praznog foldera.

7.8.5 cmdMd

Ispitujemo da li postoji folder sa istim nazivom korišćenjem metode $exists$ objekta klase $SdFat32$ i ukoliko postoji vraćamo odogovarajuću grešku. U suprotnom pokušavamo da napravimo folder korišćenjem metode $mkdir$ nad objektom klase $SdFat32$.

7.8.6 cmdCrc

Obrađujemo argumente. Prvi argument predstavlja naziv fajla i ostali argumenti su opcionalni. Prilikom uspešnog izvršavanja računamo CRC16 vrednost za segment fajla i vraćamo odgovarajuću poruku.

7.8.7 cmdGet

Obrađujemo argumente i prilikom uspešnog validiranja argumenata čitamo iz fajla odgovarajući broj karaktera. Za pročitani karakter uzimamo njegovu celobrojnu vrednost i šaljemo je preko serijskog porta. Na kraju ukoliko je prosleđen i opcionalni argument [”crc”] prosleđujemo i CRC16 vrednost izračunatu za pročitani blok.

7.8.8 cmdCd

Obrađujemo argumente. Ukoliko prvi argument predstavlja:

1. "/" ; vraćamo se na *root* korišćenjem metode $chdir$ objekta klase $SdFat32$. Takođe i objekat wp klase $CwdPath$ resetujemo.
2. ".." ; izbacujemo iz lanca celokupne radne putanje zadnji folder. Apsolutnu putanju trenutnog radnog direktorijuma čuvamo u objektu sa nazivom wp klase $CwdPath$. Nakon toga korišćenjem metode $chdir$ nad objektom klase $SdFat32$ sa argumentom $wp.get()$ menjamo trenutni radni direktorijum.
3. naziv direktorijuma ; pokušavamo da pređemo na zadati folder. U slučaju greške vraćamo odgovarajuću poruku.
4. prazan agrument odnosno ukoliko prvi argument ne postoji vraćamo naziv trenutnog radnog direktorijuma.

7.8.9 errorCodes.h

U ovom fajlu navodimo kodove grešaka.