

SVEUČILIŠTE U ZAGREBU  
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

**SEMINAR**

**Proučavanje sustava za izradu izvršnog koda na  
višestrukim platformama**

*Maja Ovčarik*

*Voditelj: Hrvoje Džapo*

Zagreb, travanj, 2017.

## Sadržaj

1. Uvod.....	3
2. Unutrašnji program za inicijalizaciju sustava .....	4
2.1 Razvojni sustav Arduino.....	5
3. Kreiranje izvršnog programa (engl. build).....	6
3.1 Programsko prevođenje .....	7
3.2 Povezivanje (engl. linking) .....	7
4. Makefile datoteka .....	10
5. Platforme za izvršni kod .....	12
6. Zaključak .....	14
7. Literatura .....	15
8. Sažetak .....	16

## 1. Uvod

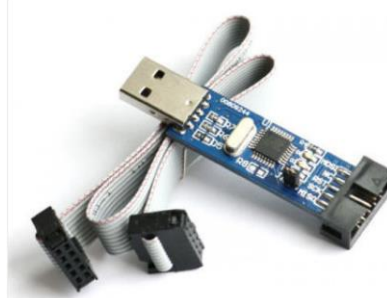
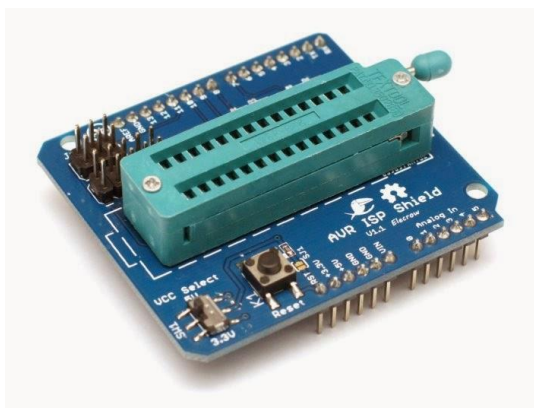
U današnje vrijeme programiranje jednostavnog razvojnog sustava poput Arduina se pokušava što više popularizirati među najmlađim generacijama. U znaku toga, ideja je napraviti aplikaciju za grafičko programiranje istog tog razvojnog sustava. Osim programiranja Arduina s prijenosnog računala, poželjno je omogućiti programiranje i s drugih platformi poput tableta ili mobitela, kao i raznih operacijskih sustava Android, iOS ili Windows.

Jedan od problema koji se javlja pri realizaciji ovakve aplikacije je izrada izvršnog programa i prijenos istog na razne platforme i operacijske sustave. Ovaj seminarski rad se bavi istraživanjem optimalnog rješenja najbržeg prevođenja izvornog koda. Kroz ovaj seminarski rad, omogućeno je upoznavanje s temeljnim pojmovima potrebnim za razumijevanje konačnog rješenja.

U prvom poglavlju se govori konkretno o programu za učitavanje operacijskog sustava (engl. *bootloader*). U drugom poglavlju se može naći sve o programskom prevodiocu (engl. *compiler*) kojeg koristi isti razvojni sustav, dok se u idućem poglavlju detaljno opisuje izrada *makefile*-ova. U zadnjem poglavlju su objašnjene razlike koje se pojavljuju pri prijenosu izvršnog programa na razvojni sustav, u ovom slučaju Arduino, s različitih platformi poput Android-a, Linuxa ili Windowsa. Dok će u zaključku biti navedeno konačno rješenje za što jednostavniju realizaciju višeplatformskog prevođenja.

## 2. Unutrašnji program za inicijalizaciju sustava

Mikrokontroler pri svom paljenju zna samo jednu stvar, a to je pokretanje instrukcija sa specifične memorijske lokacije. Ta specifična memorijska lokacija se zove početna adresa, ona najčešće sadržava asemblersku instrukciju za skok na memorijsku lokaciju na kojoj se nalazi korisnikov program. Naime, unutrašnji program za inicijalizaciju sustava (engl. *bootloader*) se nalazi u posebnom dijelu memorije, odvojenom od korisnikovog programa. Nakon prvog uključivanja mikrokontrolera ili reseta, unutrašnji program za inicijalizaciju sustava se izvršava prije izvršenja korisnikovog programa. Koristi se kako bi inicijalizirao sučelja za komunikaciju s računalom na kojem je izvršni program, uspostavlja vezu s istim, učitava sadržaj s računala i upisuje ga u *flash* memoriju razvojnog sustava. Unutrašnji program za inicijalizaciju sustava također može onemogućiti naknadnu konfiguraciju korisnikova programa. Program za učitavanje sustava je zapravo jedan od načina za vrlo lagan prijenos izvršnog koda na bilo koji uređaj. Drugi način prijenosa se obavlja preko posebnog sklopovlja koji se zove programator. Programator paralelno ili serijski prenosi izvršnu datoteku na mikrokontroler.



Slika 2-1. Programatori

Pri korištenju ovakvog načina programiranja, memorija u koju se sprema korisnikov program je veća jer ju ne zauzima program za učitavanje sustava.

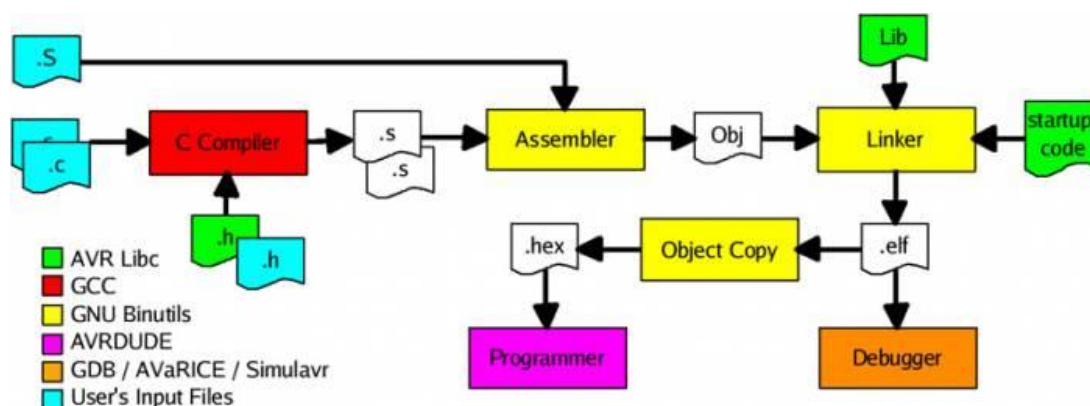
## 2.1 Razvojni sustav Arduino

Unutrašnji program za inicijalizaciju sustava kod razvojnog sustava Arduino omogućava reprogramiranje *flash* memorije preko serijske komunikacije. Unutrašnji program za inicijalizaciju na Ardiunu nakon paljenja ima određeno vrijeme čekanja, u kojem osluškuje izvode UART-a. U slučaju da unutrašnji program za inicijalizaciju ne primi predefiniranu poruku (niz bitova) preko serijskog sučelja, tada procesor napravi skok na memorijsku adresu na kojoj se treba nalazi početak korisnikove izvršne datoteke i učitava bilo kakvu datoteku koja se već nalazi u *flash* memoriji. Ako unutrašnji program za inicijalizaciju primi predefiniranu poruku tada Arduinov unutrašnji program za inicijalizaciju postaje AVR programer. AVR je familija mikrokontrolera koju je proizveo Atmel, temelje se na 8-bitnoj RISC arhitekturi, ovi mikorkontroleri su bili prvi koji su koristili *flash* memoriju za spremanje izvršnog programa. U ovoj točki procesa prebacivanja izvršne datoteke zapravo počinje slanje binarnog podataka s posebnim protokolom, koji ovisi o vrsti Arduina, preko serijskog izvoda. *Flash* memorija mikrokontrolera sprema dolazne binarne podatke, nakon završetka primanja podataka Arduino se resetira i započinje izvršavanje prenesene izvršne datoteke. Dakle, preprogramiranje *flash* memorije je moguće samo uz određeni programator ili uz unaprijed programiran unutrašnji program za inicijalizaciju sustava. Ideja je spriječiti ili onemogućiti zlonamjerne ili loše kodove od mijenjanja sadržaja *flash* memorije.

Osim mnogo prednosti koje pruža program za inicijalizaciju sustava, postoje i mnogi nedostaci. Jedan od glavnih nedostataka je kašnjenje koje unosi takav program pri pokretanju korisnikovog koda, osim toga on zauzima ionako mali memorijski prostor. Kako se taj program izvodi prije samog izvršnog koda, može postaviti neke određene registre na krivu vrijednost, ali u današnje vrijeme većina takvih problema je riješena konstantnim prepravljanjem programa za učitavanje, koji uobičajeno dolazi unaprijed programiran na svaki Arduino. Uzevši u obzir sve prednosti i nedostatke unutrašnjeg programi za inicijalizaciju Arduino sustava, da se zaključiti je da takvi programi nisu savršeni, ali uvelike olakšavaju programiranje te samim time približavaju mikrokontrolere poput Arduina široj publici.

### 3. Kreiranje izvršnog programa (engl. *build*)

Kada programeri pričaju o izradi raznih programa, često se spominje programsko prevođenje (engl. *compile*) u kontekstu iz kojega se zaključuje da programski prevodilac radi izvršnu datoteku. Takav zaključak je pogrešan, jer programsko prevođenje nije isto što i stvaranje izvršne datoteke. Stvaranje izvršne datoteke je proces koji se sastoji od dvije glavne komponente: programskog prevođenja (engl. *compilation*) i povezivanja (engl. *linking*). U stvarnosti, čak i ako pri prevođenju programa ne postoji pogreška, takav program zapravo ne mora nužno raditi jer može nastati pogreška pri povezivanju. Zbog toga, proces programskog prevođenja i povezivanja se točnije naziva kreiranje izvršnog programa (engl. *build*).



Slika 3-1. Proces kreiranja izvršnog koda.

Na slici 3-1. *AVR Libc* predstavlja biblioteku AVR od tvrtke Atmel, koja definira imena registara s njihovim adresama, kao i ostale korisne funkcije. GCC s GNU *binutils* generira *.hex* datoteke. *Avrdude* služi za programiranje memorijskih lokacija koje se nalaze u *.hex* datoteci. *AVR-GDB*, *AVaRICE*, *SimulAVR* su programi za simulaciju i ispravljanje grešaka izvršnog programa na osobnom računalu.

### 3.1 Programsko prevođenje

Programsko prevođenje se odnosi na proces prevođenja izvornog koda s ekstenzijama `.c`, `.cc` ili `.cpp`, u slučaju Arduina, u objektnu datoteku. Ekstenzije izvršnog koda označavaju da je pisan u programskom jeziku C ili C++. Arduino koristi *avr-gcc* programski prevodilac. GCC je kratica od *GNU Compiler Collection*, to je vrlo fleksibilan prevodilački sustav. Sadrži različita prevodilačka sučelja za različite programske jezike, također ima mnogo pozadinskih sustava koji generiraju asemblerski kod za razne procesore i operacijske sustave. U GCC-u glavni (engl. *host*) sustav (procesor/OS) je onaj na kojem je pokrenut prevodilac, dok je ciljani sustav (engl. *target*) onaj za koji prevodilac prevodi izvorni kod u izvršni. Prevodilac čiji je ciljani i glavni sustav jednak naziva se engl. *native compiler*, a ako su različiti poznat je pod nazivom višepatformski prevodilac (engl. *cross compiler*). U ovom slučaju glavni sustav može biti Linux, Windows ili Android, a ciljani sustav je AVR mikrokontroler, takav višepatformski prevodilac se naziva *avr-gcc*. GCC je koncentriran na prevođenje visokog programskog jezika u asemblerski kod ciljanog sustava, dok je poveziivač i sam assembler dio drugog programa zvanog GNU *binutils*. GCC se često naziva „upravljačkim“ programom, jer on daje instrukcije GNU *binutils*-u koji se sastoji od GNU *linker*-a i GNU *assembler*-a za daljnje povezivanje i prevođenje u svrhu dobivanja konačne izvršne datoteke.

### 3.2 Povezivanje (engl. *linking*)

Povezivanje se odnosi na stvaranje izvršne datoteke iz više objektnih datoteka. U ovom koraku, često se događa da će poveziivač javljati greške o nedefiniranim funkcijama. Tijekom prevođenja, ako prevodilac ne može pronaći definiciju određene funkcije, on pretpostavi da je definirana u nekoj drugoj datoteci. U slučaju da funkcija zaista nije definirana, prevodilac to neće znati jer on u određenom trenutku vidi sadržaj samo one datoteke u kojoj se nalazi program koji prevodi, dok poveziivač traži referencu funkcije u svim dostupnim datotekama. Kada se izlazna datoteka projekta treba posložiti od većeg broja datoteka, onda se povezivanje radi

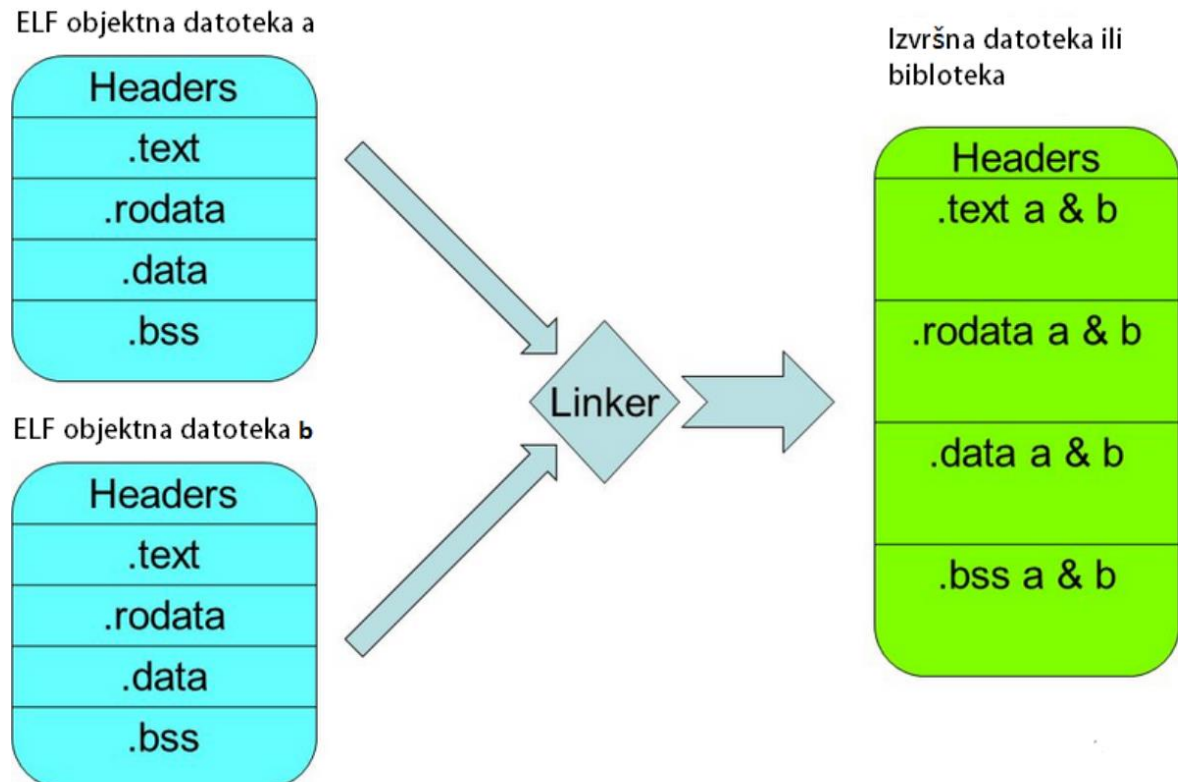
povezivanjem svih tih izlaznih datoteka. Kada se radi o običnom programu, poveziivač će sam posložiti odjeljke datoteka prema uobičajenim pravilima (sve instrukcije zajedno, konstante zajedno i slično). Međutim, kada se izgrađuje program za ugrađeno računalo tada je često potrebno definirati na kojim će se adresama nalaziti pojedini dio programa. Upute kako posložiti izlaznu datoteku iz ulaznih opisuje se posebnom datotekom koja se koristi pri povezivanju, koja se kod Arduina naziva *avr-ld*. Glavni dio takve datoteke su definicije izlaznih odjeljaka. Svaka objektna datoteka se sastoji od odjeljaka (sekcija). Odjeljci su imenovani različitim imenima, ali prevoditelji uglavnom uvijek koriste ista imena. Uobičajeni naziv odjeljka za instrukcije je *.text*, za konstante *.rodata*, *.data* za globalne varijable te *.bss* za dinamičke podatke (stog, neinicijalizirane globalne varijable itd.). Svaki odjeljak objektna datoteke definira što ide u njega i gdje se treba učitati da j odjeljak pri pokretanju te za koju adresu se odjeljak priprema. Primjer odjeljka:

```
/* unutar SECTIONS dijela */
ime_odjeljka [adresa za koju se priprema] : [AT(adresa na koju će se učitati)]
{
    datoteke_1 ( odjeljci_1 )
    [datoteke_2 ( odjeljci_2 )]
    [...]
}
```

Ako se radi o varijablama koje će se početno nalaziti na jednoj memorijskoj lokaciji, a pri pokretanju koda se treba prebaciti na drugu memorijsku lokaciju i s te iste se koristiti u daljnjem izvršavanju programa, tada će [AT(adresa na koju će se učitati)] biti adresa prve memorijske lokacije, dok će [adresa za koju se priprema] biti adresa druge memorijske lokacije. *Datoteke\_1* je popis datoteka ili filtra koji može uključivati jednu ili više objektnih datoteka, dok u *odjeljci\_1* može biti popis imena odjeljaka koji će se iz tih datoteka uzimati ili filter koji uključuje više njih (npr. *.\*data\**). Ovakav način kreiranja izvršnog programa je odličan, jer je prvenstveno lakše implementirati stvari na ovakav način. Odvajanjem procesa prevođenja i povezivanja, smanjena je kompleksnost cijelog programa te su veće šanse za brži pronalazak grešaka u kodu. Drugi razlog je taj što omogućava izradu velikih programa bez višestrukog prevođenja već prevedenih ne promijenjenih datoteka, potrebno je ponovno prevesti samo one datoteke koje su modificirane od vremena zadnjeg prevođenja, te tada samo povezati nove, upravo prevedene,



objektne datoteke i one već prije prevedene. Alat za praćenje promjene i vremena promjene datoteka na operacijskim sustavima koji koriste komandu liniju zove se *make*. *Make* se koristi za definiranje koje datoteke treba ponovno prevesti, na temelju definiranih kriterija. Pri korištenju programske podrške Arduino IDE (engl. *integrated development environment*) nije potrebno pisati make datoteke jer je taj problem riješen unutar programske podrške.



Slika 3-2. Proces povezivanja

## 4. *Makefile* datoteka

Alat *Make* prati vrijeme promjena datoteke te pri prevođenju prevodi samo one koje su se promijenile od zadnjeg prevođenja. Postupak cjelokupnog prevođenja velikih projekata se znatno ubrzava ponovnim prevođenjem samo promijenjenih datoteka, što je ujedno i jedan od glavnih razloga korištenja *Makefile*-a. Pri prevođenju projekata koriste se tri glavna pristupa smještaja objektnih datoteka. Prvi i najjednostavniji pristup ostavlja prevedene datoteke uz izvorne (u istim direktorijima). Nedostatak ovog pristupa je što gomila datoteke u direktorijima s izvornim kodovima te time smanjuje preglednost. Prema drugom pristupu sve se prevedene datoteke smještaju u jedan zaseban direktorij. Problem s ovim pristupom je u imenima datoteka: ako postoje dvije datoteke istog imena u različitim direktorijima onda će prevođenje druge prebrisati prvu prevedenu. Rješenje tog problema je proširivanje imena datoteke cijelom njenom putanjom. Prema trećem pristupu u svaki se direktorij s izvornim kodom postavlja datoteka *Makefile* koja sadrži upute kako te datoteke prevesti. Iz početnog se *Makefile*-a tada spušta po direktorijima i pokreće prevođenje na lokalnoj razini. Ovaj pristup omogućava znatno više slobode pri prevođenju i često se koristi za veće projekte. Jednostavni *Makefile* sastoji se od pravila koja su uvijek u sljedećem obliku:

```
Ciljana datoteka... : preduvjeti ...  
  
    recept  
    ...  
    ...
```

Ciljana datoteka (engl. *target*) je uobičajeno ime datoteke koju će generirati program, npr. izvršna datoteka ili objektna datoteka. Ciljani sustav može biti i ime naredbe (npr. *clean*) koja se izvršava odgovarajućim pozivom. Preduvjeti su ulazne datoteke koje se koriste pri stvaranju ciljane datoteke. Ciljana datoteka često ovisi o više različitih datoteka, koje trebaju biti navedene iza dvotočke prvog retka. Recept je funkcija koju alat *make* izvodi. Recept može sadržavati više redaka, oni obavezno moraju biti uvučeni. On se može promatrati kao skup pravila koja služe za stvaranje

ciljane datoteke u slučaju promjene bilo koje datoteke navedene u preduvjetima. Naime, pravila koja čine recept za ciljanu datoteku ne moraju nužno imati preduvjete. Primjer takve iznimke je ciljana naredba *clean*, koja pozivom iz naredbenog retka direktno izvršava svoj recept iako ne sadrži ništa u preduvjetima. Pravilo nalaže kada i kako prepraviti određene datoteke koje su ciljana datoteka određenog pravila. Alat *make* izvršenjem recepta, temeljen na potrebnim preduvjetima, stvara ili ažurira ciljanu datoteku. Nadalje, svaki *makefile* može sadržavati i druge dijelove teksta osim ovih već spomenutih koji su temelj svake takve datoteke. U nastavku je demonstriran primjer jednostavne *makefile* datoteke.



```
hellofun.c
#include <stdio.h>
#include "hellomake.h"

void myPrintHello(void)
{
    printf("Hello world!\n");
    return;
}

hellomake.c
#include "hellomake.h"

int main()
{
    myPrintHello();
    return 0;
}

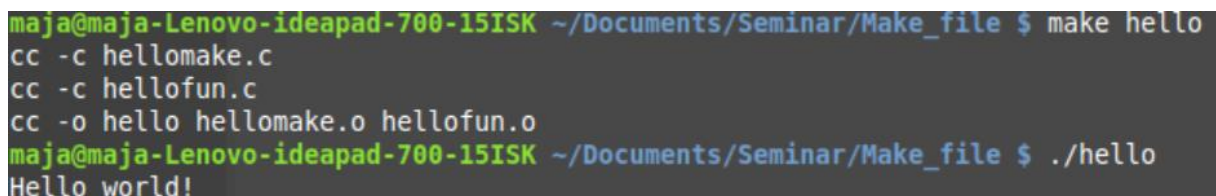
hellomake.h
void myPrintHello(void);

makefile
hello : hellomake.o hellofun.o
    cc -o hello hellomake.o hellofun.o

hellomake.o : hellomake.c hellomake.h
    cc -c hellomake.c

hellofun.o : hellofun.c hellomake.h
    cc -c hellofun.c

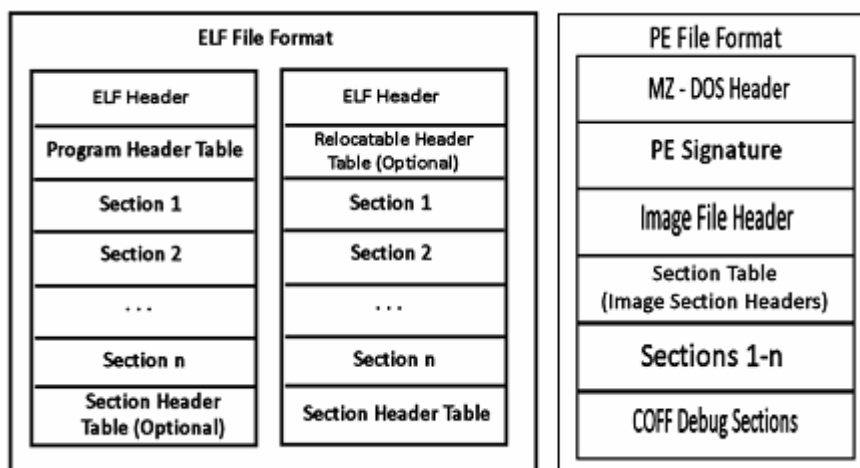
clean:
    rm hello| hellomake.o hellofun.o
```



```
maja@maja-Lenovo-ideapad-700-15ISK ~/Documents/Seminar/Make_file $ make hello
cc -c hellomake.c
cc -c hellofun.c
cc -o hello hellomake.o hellofun.o
maja@maja-Lenovo-ideapad-700-15ISK ~/Documents/Seminar/Make_file $ ./hello
Hello world!
```

## 5. Platforme za izvršni kod

U poglavlju broj tri ustanovljeno je da je ciljani sustav za izvršni kod AVR mikrokontroler, dok glavni sustav može biti Android, operacijski sustav Windows ili operacijski sustav Linux. Između operacijskih sustava Windows i Linux ima mnogo razlika. Svaki operacijski sustav zahtjeva drugačiju izvršnu datoteku koja na svim operacijskim sustavima sadrži jedan isti dio – izvorni kod. Ostatak izvršne datoteke je popunjen statičkim i dinamičkim podacima te informacijama za povezivanje i bibliotekama koje ovise od sustava do sustava. Na operacijskom sustavu Linux izvršna datoteka je spremljena u `.elf` formatu (engl. *ELF – executable and linkable format*), dok je na operacijskom sustavu Windows spremljena u formatu `.exe`. Dinamički podaci na Windowsu su spremljeni u datoteke s ekstenzijom `.dll`, dok su na operacijskom sustavu Linux implementirane kao dijeljenje datoteke s ekstenzijom `.so`. Iz navedenih razloga jedna izvršna datoteka se ne može pokrenuti na oba operacijska sustava, nego je istu potrebno ponovno prevoditi i povezati te tako stvoriti odgovarajuću izvršnu datoteku za oba sustava.



Slika 5-1. Prikaz sadržaja izvršnih datoteka za Windows i Linux

Povezivanje ciljanog sustava za izvršni kod AVR mikorontrolera – Arduina te njegovo programiranje moguće je izvesti preko raznih tableta i mobitela čiji je operacijski sustav Android. Potrebno je Android uređaj postaviti u *USB host mode*.

Kada je uređaj u takvom načinu rada, ponaša se kao osobno računalo, odnosno kao glavni sustav, dok je priključeni uređaj ciljani sustav. USB *host* način rada je podržan od Andorid 3.1 pa nadalje. Kod uređaja koji ne podržavaju ovakva način rada, program se može prebaciti putem *Bluetootha*. Osim toga, potrebno je napisati popratnu programsku podršku koja ostvaruje sam prijenos datoteke, odnosno modificirati postojeći poveziivač i prevoditelj. Detaljan opis modifikacije programske podrške prelazi okvire ovoga rada.

## 6. Zaključak

Kako bi prebacili korisnikov kod na Arduino sustav potrebno je proći dug put, koji se sastoji od mnogo ključnih koraka. Nakon generiranja blokovske sheme u napravljenoj aplikaciji, potrebno je odlučiti na koji način će biti izvedeno prevođenje i povezivanje. Moguća su dva načina, prvi način je taj da aplikacija iz blokovske sheme generira cjelokupan kod u programskom jeziku C ili C++, te taj generirani kod prosljeđuje programskom prevodiocu u ovom slučaju *avr-gcc*-u, koji prebacuje izvorni kod u objektnu datoteku. Tada sustav za povezivanje (engl. *linker*) povezuje objektnu datoteku sa standardnim bibliotekama Arduina koje omogućuju korištenje osnovnih funkcija poput *digitalWrite()* ili *Serial.print()*. Drugi način je takav da postoje već prevedene objektna datoteke za svaku blokovsku naredbu, te kao takve budu dio jedne biblioteke. U tom slučaju bi aplikacija trebala generirati kod koji povezuje blokove, te ih proslijediti programskom prevodiocu koji zatim stvara objektnu datoteku. Povezivač ju tada povezuje sa standardnim bibliotekama Arduina kao i dodatnom bibliotekom u kojoj se nalaze objektna datoteke blokovskih naredbi.

U oba načina izlaz iz sustava za povezivanje je jedna izvršna datoteka (često s nastavkom *.hex*) koja sadrži pre-definiranu poruku koja treba biti zapisana u *flash* memoriju Arduino sustava. Nakon što se u *flash* memoriju prebaci izvršna datoteka putem serijske komunikacije i unutrašnjeg programa za inicijalizaciju, USB-a ili sklopovlja za programiranje, mikrokontroler se resetira te počinje izvršavanje prenesene datoteke, odnosno napisanog korisnikovog programa.

Drugi način prevođenja i povezivanja je brži nego prvi način, zbog toga što je veliki dio cijele sheme već preveden i prebačen u objektnu datoteku, ali je potrebno napisati novi povezivač jer povezivač u AVR-GCC lancu tako ne funkcionira. Dok s druge strane gledano, prvi način prenošenja programa je sporiji, ali radi uz unaprijed gotov prevoditelj i povezivač. Također, aplikacija na raznim operacijskim sustavima mora biti promijenjena sukladno zahtjevima sustava, dok pozadinska programska podrška koja se sastoji od povezivača i prevoditelja uz manje modifikacije jednako za sve sustave.

## 7. Literatura

- [1] *Joseph Yiu – „The definitive guide to the ARM CORTEX-M3“*, 2009.
- [2] *Leonardo Jelenković – „Operacijski sustavi za ugrađena računala“*, 2016.
- [3] „GNU make“, <https://www.gnu.org/software/make/manual/make.html#Reading>
- [4] „AVR Libc Reference Manual Toolchain Overview“, [http://www.atmel.com/webdoc/avrllibcreferencemanual/overview\\_1overview\\_gcc.html](http://www.atmel.com/webdoc/avrllibcreferencemanual/overview_1overview_gcc.html), 2017

## 8. Sažetak

Popularizacija znanosti, tehnologije, inženjerstva i matematike je jedna od vrlo bitnih stavki današnjeg društva u kojemu nedostaje stručnih ljudi u navedenim područjima, te se ona sve češće provodi već u najmlađim generacijama. Arduino je jedan od najraširenijih mikrokontrolera današnjice, uz to je dostupan u vrlo niskom cjenovnom rangu te je također vrlo prilagođen korisniku za što lakšu i bržu upotrebu. Uz sve navedene pogodnosti, najmlađe generacije je vrlo lako zaintrigirati s raznobojnim svjetlosnim indikatorima koji se pomoću odgovarajuće programske podrške i Arduina vrlo lako mogu programirati, te im tako približiti i razviti drugačiji način razmišljanja i usmjeriti ih na prednosti STEM područja. Od grafičkog sučelja do samog izvršavanja koda na mikrokontroleru, potrebno je proći mnogo procesa. Procesi poput prevođenja datoteka, povezivanja prevedenih datoteka s ugrađenim bibliotekama te konačnog prebacivanja izvršnog programa na mikrokontroler zahtijevaju mnogo složene pozadinske programske podrške, čija je izvedba opisana detaljno u ovom seminarskom radu.