Vežba 3 : Aplikacije za rad sa drajverima u Linuxu

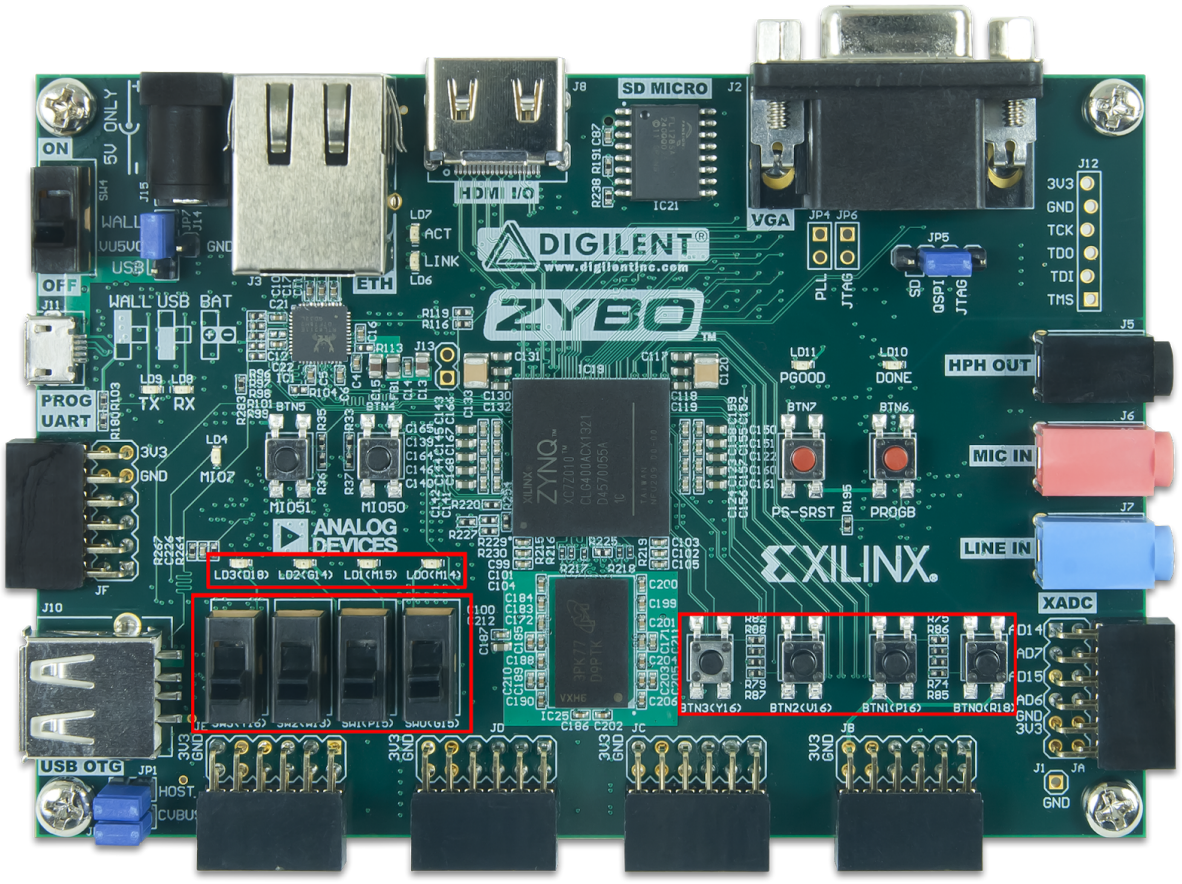
1. Drajveri u Linux-u

U Linux-u korisnik može pristupiti hardverskim uređajima kroz specijalne “node” fajlove. Ovi fajlovi se nalaze u direktorijumu /dev, i svakom uređaju odgovara tačno jedan fajl. Kada se nad ovakvim fajlom izvrši sistemski poziv kao što je: open, read, write, close, lseek ili mmap, operativni sistem taj poziv preusmeri u drajver za odgovarajući uređaj. Drajver oplužuje ovaj poziv i dalje komunicira sa uređajem. Prethodno pomenuti sistemski pozivi su namenjeni za rad sa tekstualnim fajlovima pa se sa ovakvim uređajima komunicira čitanjem ili upisom stringa u njihov “node” fajl. Drajveri za karakter uređaje će biti detaljno opisani u vežbama koje slede, dok ćemo se na ovim vežbama zadržati samo na komunikaciji sa njima.

Na slici 1 je prikazana Zybo razvojna ploča, gde su crvenim pravougaonicima uokvirene 4 LED (LD0-LD3), 4 prekidača (SW0-SW3) i 4 tastera (BTN0-BTN3). Za ove uređaje su napisani drajveri i u direktorijumu /dev se mogu primetiti sledeća tri “node” fajla: led, button i switch. Kako bismo komunicirali sa drajverom moramo znati koje sve komande on podržava i u kakvom ih formatu očekuje. Za ozbiljnije uređaje bi to bilo detaljno opisano u dokumentaciji, dok u našem slučaju za time nema potrebe.

LED se uključuju upisom vrednosti u fajl “/dev/led”. Vrednost u stringu može biti predstavljena u binarnom (0b1010), heksadecimalnom (0xA) ili dekadnom formatu (10). U kojem god formatu da se vrednost prosledi, biće korištena samo najniža 4 bita, te će svaki bit predstavljati vrednost jedne od dioda. Ukoliko bit ima vrednost ‘1’ dioda će se upalit, dok će se ugasiti za vrednost ’0’. Npr. komanda: **echo “0b1010” > /dev/led** će upaliti diode LD1 i LD3 dok će ugasiti diode LD0 i LD2 (videti sliku 1). Komanda **echo “0xA” > /dev/led** je ekvivalentna prethodnoj. Stanje LED se može i pročitati komandom: **cat /dev/led** čime će se u terminalu u binarnom zapisu ispisati koje su diode uključene. Npr. **cat /dev/led** rezultuje ispisom **0b1111** što znači da su sve četiri diode upaljene.

Sa tasterima i prekidčima se komunicira na sličan način kao i sa LED, s razllikom da se u tastere i prekidače ne mogu upisati vrednosti već se samo može pročitati njihovo stanje. Kada se pročitaju vrednosti tastera, biti na kojima je očitana nula su oni koji su trenutno pritisnuti. Npr. ukoliko naredba **cat /dev/button** rezultuje u ispisu **0b1110**, to znači da je taster BTN0 pritisnut. Prekidači su ekvivalentni tasterima, s tim što imaju dva položaja (0 i 1). Vrednost prekidača se može preuzeti čitanjem fajla **/dev/switch**.



Slika 1. Zybo ploča

Komunikacija sa drajverima u terminalu koristeći naredbe echo i cat je često korisna za ispitivanje da li su drajver i uređaj prisutni i da li korektno rade, ali se uređaji u većini slučajeva koriste u sklopu nekog kompleksnog C programa. Iz ovog razloga će u sledećem poglavlju u kratkim crtama biti opisan način na koji se iz C aplikacije komunicira sa ulazno-izlaznim (I/O) podsistemom i biće navedene često korištene funkcije za manipulisanje stringovima.

2. Funkcije za manipulisanje stringovima i I/O podsistemom

2.1. Tokovi (Stream)

Datotekama se u linuksu pristupa na jedan od dva načina:

- Na visokom nivou - pomoću tokova (stream)
- Na niskom nivou - pomoću file deskriptora (file id)

Oba načina za manipulisanje datotekama omogućavaju osnovne operacije kao što su otvaranje, zatvaranje, upis, čitanje itd. Preporučuje se upotreba tokova zato što poseduje moćnije alate (funkcije) za formatiranje upisa i ispisa. Najpoznatiji tokovi u linuksu su *stdout* i *stdin* preko kojih se omogućuje ispisivanje poruka u terminalu kao i unos informacija od strane korisnika (*printf, scanf*).

2.2. Otvaranje datoteke, kreiranje toka

Pre nego što se pročita sadržaj datoteke, ili se u nju upiše neka vrednost, neophodno je otvoriti datoteku. Za ovu svrhu se koristi komanda **fopen** koja vraća tok (stream). Tok je predstavljen pokazivačem na strukturu FILE, te se nov tok deklariše pomoću:

FILE* my_stream;

Kada se otvori datoteka, GNU C biblioteka kreira novi tok i konekciju sa datotekom. Ukoliko se ovoj funkciji prosledi naziv datoteke koja ne postoji, ona će biti kreirana. Fopen funkcija vraća tok koji je kreiran prilikom uspostavljanja konekcije sa datotekom.Prvi parametar funkciji je string koji sadrži naziv datoteke koja se otvara, dok je drug parametar mod u kojem se datoteka otvara:

My_stream = fopen(“/root/primer.txt”,”r”)

Modovi koji su podržani su:

- **r** Otvori datoteku samo za čitanje. Datoteka mora da postoji.
- **w** Otvori datoteku samo za upisivanje. Ako datoteka postoji, njen prethodni sadržaj će biti obrisan, a ukoliko ne postoji, biće kreirana.
- **r+** Otvori datoteku za čitanje i upis. Datoteka mora da postoji. Sadržaj datoteke se inicijalno ne menja, ali je pozicija unutar datoteke postavljena inicijalno na početak.
- **w+** Otvori datoteku za čitanje i upis. Ukoliko datoteka postoji, njen prethodni sadržaj će biti obrisan, a ukoliko ne postoji, datoteka će biti kreirana.

- **a** Otvori datoteku samo za dodavanje sadržaja, što je isto kao i upis u datoteku, sa razlikom da se novi podaci upisuju samo na kraj već postojeće datoteke. Ako datoteka ne postoji, biće kreirana.
- **a+** Otvori datoteku samo dodavanje sadržaja i čitanje. Ako datoteka postoji, njen sadržaj se neće menjati sve dok se ne doda novi sadržaj, a ukoliko ne postoji, biće kreiran. Inicijalna pozicija za čitanje unutar datoteke je na početku datoteke, ali je pozicija za dodavanje na kraju datoteke.
- Moguće je dodati karakter x nakon svakog karaktera iz tabele navedene iznad. Ovaj karakter podrazumeva neuspešno korišćenje fopen funkcije ukoliko datoteka već postoji i na ovaj način se štitimo od slučajnog brisanja datoteke prilikom otvaranja datoteke.

2.3. Zatvaranje datoteke, prekidanje toka

Funkcija koja se koristi za zatvaranje datoteke je **fclose**, kojoj se prosleđuje tok a ona će prekinuti vezu sa odgovarajućom datotekom. Ukoliko je datoteka uspešno zatvorena, vraća se vrednost 0, u suprotnom vraćena vrednost je EOF.

```
fclose(my_stream);
```

2.4. Pozicija u datoteci, indikatori kraja i greške

U slučaju rada sa datotekama je potrebno znati poziciju do koje se stiglo prilikom čitanja ili upisivanja u datoteku. Takođe, često može biti korisno imati mogućnost promene trenutne pozicije u okviru datoteke. Funkija **ftell**(my_stream) kao jedini parametar prima tok a vraća trenutnu poziciju u datoteci. Funkcija **fseek**(my_stream,offset,where) menja poziciju u fajlu za “offset” bajtova (karaktera) u odnosu na “where”. Konstanta “where” specificira da li je ofset relativan u odnosu na početak datoteke (SEEK_SET), trenutnu poziciju (SEEK_CUR), ili kraj datoteke (SEEK_END). Fseek funkcija vraća vrednost 0 ukoliko je uspešno obavljena ova operacija, ili nenultu vrednost u suprotnom. Funkcija **rewind**(my_stream) vraća kursor na početak fajla. Funkcija **feof**(my_stream) vraća vrednost true ko se došlo do kraja datoteke. Indikator greške, **ferror**(my_stream) se koristi da signalizira da se greška pojavila prilikom prethodne operacije na toku. Ovaj indikator ima vrednost *true* ako je greška prisutna, odnosno *false* u suprotnom. Nažalost, ferror funkcija nam neće javiti koja greška je nastala ili gde je nastala, jedino da je došlo do greške.

2.5. Baferovanje

Glavna odlika tokova je da su baferovani. Niame, u FILE strukturi postoji bafer u kojem se karakteri akumuliraju, a zatim se iz bafera upisuju u datoteku u blokovima kada se određeni uslovi ispune. Ovo se radi kako bi se smanjio broj sistemskih poziva.

Postoje tri osnovna tipa baferovanja koji su od interesa: •

- Bez baferovanja - Kada se karakteri upisuju u ne-baferovani tok, operativni sistem ih upisuje u datoteku čim je to moguće
- Linijsko baferovanje - Prilikom upisa karaktera u linijski baferovan tok, operativni sistem će ih upisati u datoteku tek kada naiđe na novi red (newline karakter)
- Puno baferovanje - Kada se upisuju karakteri ovom metodom, operativni sistem upisuje podatke u datoteku u blokovima proizvoljne veličine.

Stdin i stdout tokovi su linijski baferovani, što znači da će se poruka proslediti u terminal ili preuzeti iz terminala samo ako postoji karakter za novi red '\n'. Moguće je prisilno isprazniti bafer pomoću komande **fflush**(my_stream).

2.6. Ispis stringova (print)

Funkcija **puts** je najzgodnija funkcija za ispis jednostavnih poruka na standardnom izlazu stdout. Još je jednostavnija od printf, jer nije potrebno dodavati karakter za novi red, puts to radi automatski.

```
puts ("Hello.");
```

Funkcija **fputs** ("file put string") je slična puts funkciji, osim što prihvata i drugi parametar, koji predstavlja tok u koji se upisuje string prosleđen kroz prvi parametar. Ova funkcija ne dodaje newline karakter. Vraća EOF ukoliko je došlo do greške, u suprotnom vraća nenegativnu celobrojnu vrednost.

```
fputs ("Hello again\n", my_stream);
```

2.7. Formatirani ispis (printf)

Formatirani ispis podrazumeva tekstualni izlaz preko funkcija kao što su printf ili fprintf. Ove funkcije primaju kao argument string koji sadrži specijalne sekvence karaktera (označeni sa %) koji se zamenjuju sa tekstualnom reprezentacijom argumenata.

Format za konverziju argumenata je:

%[indikatori][širina][.preciznost][dužina]specifikator

Podržani specifikatori su:

- %d,%i celobrojna vrednost sa predznakom
- %u celobrojna vrednost bez predznaka
- %o oktalna vrednost bez predznaka
- %x heksadecimalna vrednost bez predznaka
- %X heksadecimalna vrednost bez predznaka (velika slova)
- %f decimalna vrednost sa pokretnom tačkom
- %F decimalna vrednost sa pokretnom tačkom (velika slova)
- %e scientific format (mantisa + eksponent)
- %E scientific format (mantisa + eksponent) (velika slova)
- %g koristi kraću reprezentaciju %f ili %e
- %G koristi kraću reprezentaciju %F ili %E
- %c karakter
- %s string, niz karaktera
- %p adresa pointera

Širina je broj cifara, sa kojima će celobrojna vrednost biti prikazana.
Preiznost je broj cifara sa kojima će decimalni deo broja bii prikazan.
Dužina se koristi za promenu broja bita sa kojim se argument predstavlja.
Indikatori se koriste za dodatno formatiranje argumenta:

- - poravnaj broj na levu stranu
- + prikaži predznak
- [space] ako nema predznak dodaj space
- # u hex formatu dodaj predznak 0x

Funkcija **fprintf** ("file print formatted") je identična funkciji **printf**, osim što je prvi parametar tok u koji se šalje izlaz. Obe kao drugi parametar primaju formatirani string praćen argumentima.
fprintf (my_stream, "Integer: %d\n", my_integer);
printf ("Integer: %d\n", my_integer);

asprintf (mnemonic: "allocating string print formatted") je identična kao i printf, osim što je prvi parametar string u koji se šalje izlaz, umesto u tok stdout. Ova funkcija terminira string null karakterom. Vraća broj karaktera koji su smešteni u stringu, ne uključujući terminirajući null karakter. Funkcija asprintf je gotovo identična jednostavnijoj **sprintf** funkciji, ali je mnogo bezbednija za korisniccćenje, jer dinamički alocira string u kjoi šalje izlaz, tako da prilikom upisa u string nikada neće doći do prekoračenja opsega. Prvi parametar je pokazivač na string promenljivu koji je, samim tim, tipa char **. Povratna vrednost je broj karaktera koji su alocirani za bafer, ili negativna

vrednost ukoliko se desila greška. Sledeći primer pokazuje korišćenje ove funkcije (obratiti pažnju da pre poziva funkcije nije alociran prosto za niz my_string, jer će to uraditi sama funkcija asprintf):

```
char *my_string;
asprintf (&my_string, "Don’t hate m%d", 8);
```

2.8. Unos stringova

Funkcija **getline** je preferirana funkcija za čitanje ulaza, kako iz datoteka, tako i od strane standardnog ulaza. Ostale standardne funkcije koje se koriste u ovu svrhu, uključujući gets, fgets i scanf su nepouzdane. Ova funkcija čita ceo red iz toka, sve do (i uključujući) newline karakter. Ova funkcija prima tri parametra:

- pokazivač na blok alociran pomoću malloc ili calloc. Ovaj parametar je tipa char ** i sadržaće pročitanu liniju kada se funkcija izvrši do kraja.
- pokazivač na promenljivu tipa size_t koji specificira kolika je veličina u bajtovima bloka memorije na koju pokazuje prvi parametar.
- tok koji se koristi i iz koga se čitaju podaci.

```
int nbytes = 100;
my_string = (char *) malloc (nbytes + 1);
bytes_read = getline (&my_string, &nbytes, stdin);
```

Funkcija **getdelim** predstavlja generalizaciju getline funkcije; dok getline završava sa čitanjem ulaza prilikom prvog newline karaktera, getdelim funkcija omogućava korisnika da specificira neki drugi delimiter koji će se koristiti umesto newline karaktera. Zapravo, getline funkcija jednostavno poziva getdelim i specificira da je delimiter newline karakter.

2.9. Formatiranje unosa

Funkcija **sscanf** prihvata string (često preuzet sa toka pomoću funkcije getline) zatim, u maniru sličnom printf funkciji, preuzima šablon string i niz prosleđenih argumenata, pokušava da prepozna string koji se koristi kao ulaz u datom šablon stringu, pri čemu koristi konverziju isto kao u slučaju printf funkcije. Funkcija sscanf, je slična kao zastarela scanf funkcija, osim što u slučaju sscanf funkcije, na mestu prvog parametra specificira se string iz koga se čita, dok se u slučaju scanf funkcije uvek čita standardni ulaz stdin. Dostizanje kraja stringa se tretira kao i EOF uslov.

```
sscanf (input_string, "%as %as %as", &str_arg1, &str_arg2, &str_arg3);
```

2.10. Unos i ispis jednog karaktera

Ukoliko je potrebno čitati jedan karakters sa standardnog ulaza može se koristiti **getchar** funkcija.

input_char = getchar();

Ukoliko je potrebno odštampati jedan karakter na standardni izlaz, moguće je koristiti funkciju **putchar**.

putchar(output_char);

Ukoliko postoji potreba čitanja jednog karaktera iz toka drugačijeg od standardnog, moguće je koristiti **getc** funkciju. Ova funkcija je veoma slična getchar funkciji, ali prihvata argument koji specificira tok iz koga se čita.

input_char = getc (my_stream);

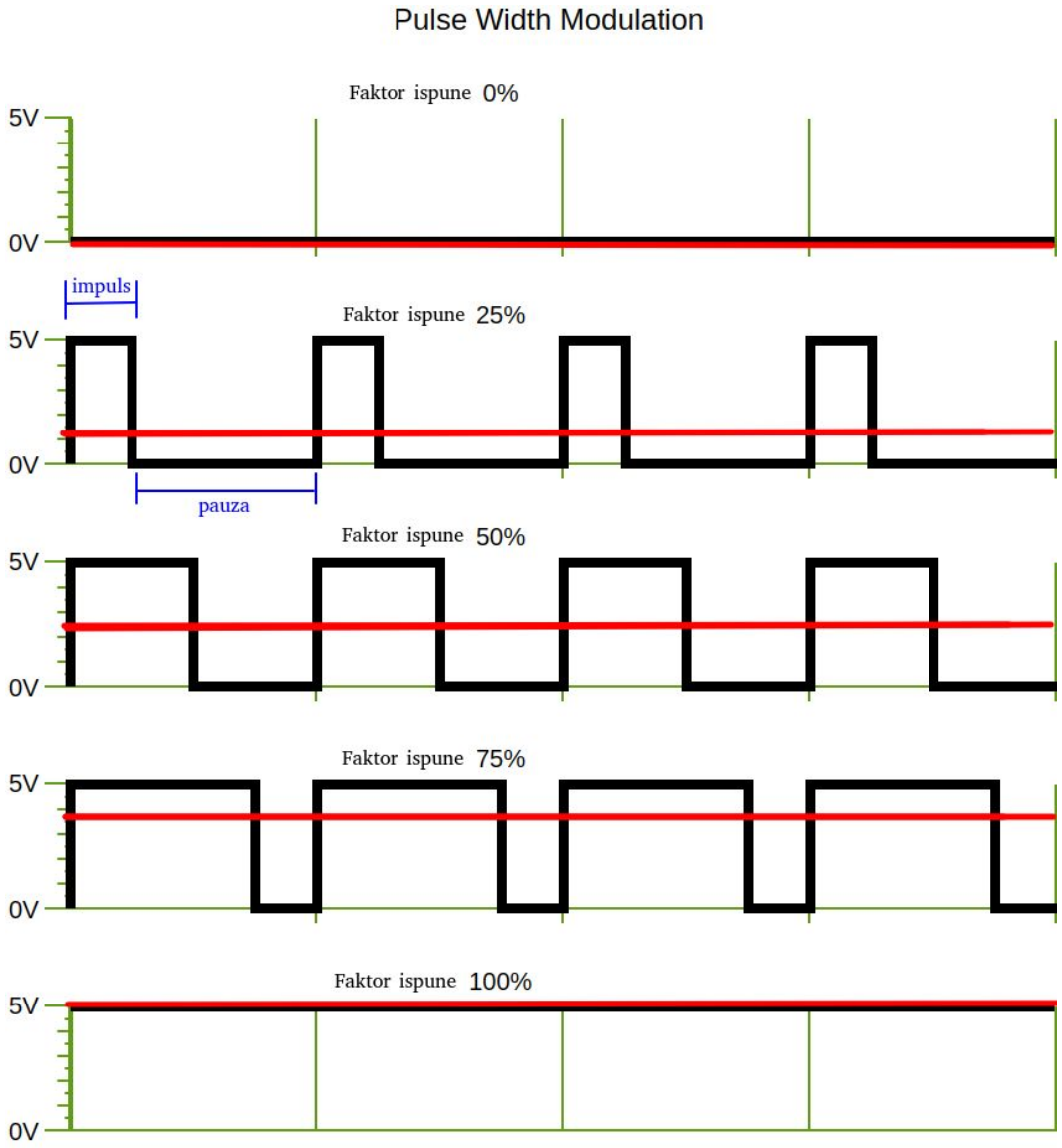
Ukoliko je potrebno ispisivati jedan karakter u tok različit od standardnog, moguće je koristiti **putc** funkciju. Ova funkcija je veoma slična putchar, ali prihvata argument koji specificira tok u koji se upisuje.

putc (output_char, my_stream);

Prilikom svakog čitanja karaktera iz toka koričćenjem funkcija kao što je getc, pozicija u datoteci se uvećava za 1. Moguće je uraditi inverznu operaciju funkcijom **ungetc**, koja vraća poziciju u datoteci nazad za jedan bajt, i poništava poslednju operaciju čitanja karaktera iz datoteke.

3. Aplikacija za rad sa diodama

Aplikacija led_pwm.c implementira impulsno širinsku modulaciju (*eng. PWM, Pulse Width Modulation*) nad 4 diode koje su prisutne na Zybo ploči. PWM je periodična povorka impulsa, gde imamo mogućnost promene širine impulsa unutar periode signala. Menjajući odnos impuls - pauza se menja faktor ispune signala. Na slici je prikazano pet slučajeva faktora ispune (0%, 25%, 50%, 75%, 100%) Ukoliko je frekvencija povorke dovoljno velika, potrošač vidi samo srednju vrednost signala (prikazanu crvenom bojom). Pomoću pwm-a je moguće sa digitalnim signalom menjati srednju vrednost signala koji se dovodi na diode, a samim tim i jačinu sa kojom one svetle.
Pri pokretanju aplikacije je neophodno da se prosledi argument u opsegu od 0 do 1 koji će predstavljati faktor ispune pwm signala (u kodu označen kao *percentage*).



Slika 2. PWM signal

Na početku glavne (main) funkcije se inicijalizuju promenljive, a zatim se proverava da li je prosleđen tačno jedan argument. Ukoliko je unet pogrešan broj argumenata ispisuje se poruka i program se terminira. U suprotnom slučaju se argument pomoću funkcije sscanf prebacuje iz stringa u vrednost sa pokretnom tačkom (float). Glavni deo programa predstavlja beskonačna petlja while(1) gde svaki prolaz kroz telo petlje predstavlja jednu periodu povorke impulsa pwm signala. Unutar beskonačne petlje se otvori fajl /dev/led u modu za upis, zatim se pomoću funkcije fputs upiše “0xF” (čime se upale led) a na kraju se fajl zatvori. Pri otvaranju i zatvaranju fajla se kao sigurnonosna mera proverava povratna vrednost funkcija fopen i fclose. Nakon što su led upaljene potrebno je da ostanu upaljene za vreme trajanja impulsa. To vreme je određeno kao (faktor ispune * perioda povorke) (videti sliku 2). Za ovo vreme program čeka pomoću funkcije usleep. Ova funkcija je definisana u biblioteci <unistd.h> i očekuje kao parametar vremenski period u mikrosekundama tokom koga će program čekati. Nakon što se program probudi, nastupa pauza u PWM signalu, te se u fajl /dev/led upisuje “0x0” a zatim se čeka ostatak periode.

```
int main(int argc, char** argv)
{
	FILE* fp;
	float percentage;
	long int period = 20000L;
	if(argc == 2)
		sscanf(argv[1],"%f",&percentage);
	else{
		printf("Pogresan broj argumenata\n");
		return -1;
	}
	while(1){
		// Upali diode
		fp = fopen("/dev/led", "w");
		if(fp == NULL){
			printf("Problem pri otvaranju /dev/led\n");
			return -1;
	}
		fputs("0x0F\n", fp);
		if(fclose(fp)){
			printf("Problem pri zatvaranju /dev/led\n");
			return -1;
		}
		usleep(percentage*period);
```

```
                // Ugasi diode
                fp = fopen("/dev/led", "w");
                if(fp == NULL){
                        printf("Problem pri otvaranju /dev/led\n");
                        return -1;}
                fputs("0x00\n", fp);
                if(fclose(fp)){
                        printf("Problem pri zatvaranju /dev/led\n");
                        return -1;}
                usleep((1-percentage)*period);
        }
}
```

4. Aplikacije za rad sa tasterima i prekidačima

Aplikacije za rad sa tasterima i prekidačima su u suštini iste, samo se čita iz drugog “node” fajla. Iz ovog razloga će biti prikazana samo jedna aplikacija. Ove aplikacije svake sekunde isčitavaju vrednosti tastera iz odgovarajućeg “node” fajla, a zatim vrednosti ispisuju u terminalu. I ova aplikacija kao glavni deo main funkcije poseduje beskonačnu petlju. Na isti način kao i kod prethodne aplikacije se otavara /dev/button fajl samo sada u modu za čitanje. Prvo se dinamički alocira memorija za string veličine 7 bajtova (6 za string + 1 za terminator stringa), a zatim se u njega smesti vrednost iz fajla pomoću funkcije getline. Budući da je završeno sa čitanjem, fajl se zatvara. Budući da je pročitana vrednost u binarnom formatu, ne trebaju nam prva dva karaktera “0b”. U vrednosti tval1,2,3 i 4 su smeštena četiri karaktera koja mogu biti ‘0’ ili ‘1’. Kako bi se karakteri iz ASCII tabele prebacili u konkretne celobrojne vrednosti potrebno je oduzeti vrednost 48. Nakon ovoga se alocirana memorija može osloboditi. Za kraj se u terminalu ispisuju preuzete vrednosti te se pomoću funkcije sleep čeka jedan sekund. Funkcija sleep se takođe nalazi u biblioteci <unistd.h> te se razlikuje od funkcije usleep po tome što očekuje sekunde kao parametar.

```
int main (){
        FILE *fp;
        char *str;
        char tval1,tval2,tval3,tval4;
        size_t num_of_bytes = 6;
```

```
while(1){
		//Citanje vrednosti tastera
		fp = fopen ("/dev/button", "r");
		if(fp==NULL) {
			puts("Problem pri otvaranju /dev/button");
			return -1;}


		str = (char *)malloc(num_of_bytes+1);
		getline(&str, &num_of_bytes, fp);

		if(fclose(fp)){
			puts("Problem pri zatvaranju /dev/button");
			return -1;}

		tval1 = str[2] - 48;
		tval2 = str[3] - 48;
		tval3 = str[4] - 48;
		tval4 = str[5] - 48;
		free(str);

		printf("Vrednosti tastera: %d %d %d %d
\n",tval1,tval2,tval3,tval4);
		sleep(1);

	}
}
```