

# Jednostavan operacijski sustav za mikroupravljača ATmega328p

---

**Tomas, Vito**

**Undergraduate thesis / Završni rad**

**2024**

*Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj:* **University of Zagreb, Faculty of Electrical Engineering and Computing / Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva**

*Permanent link / Trajna poveznica:* <https://urn.nsk.hr/urn:nbn:hr:168:827524>

*Rights / Prava:* [In copyright/Zaštićeno autorskim pravom.](#)

*Download date / Datum preuzimanja:* **2025-03-23**



*Repository / Repozitorij:*

[FER Repository - University of Zagreb Faculty of Electrical Engineering and Computing repository](#)



SVEUČILIŠTE U ZAGREBU  
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1290

**JEDNOSTAVAN OPERACIJSKI SUSTAV ZA  
MIKROUPRAVLJAČA ATMEGA328P**

Vito Tomas

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU  
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1290

**JEDNOSTAVAN OPERACIJSKI SUSTAV ZA  
MIKROUPRAVLJAČA ATMEGA328P**

Vito Tomas

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU  
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

ZAVRŠNI ZADATAK br. 1290

Pristupnik: **Vito Tomas (0036541307)**

Studij: Elektrotehnika i informacijska tehnologija i Računarstvo

Modul: Računarstvo

Mentor: doc. dr. sc. Leonardo Jelenković

Zadatak: **Jednostavan operacijski sustav za mikroupravljača ATmega328p**

Opis zadatka:

Proučiti svojstva mikroupravljača ATmega328p. Za njega osmisliti jednostavan operacijski sustav koji ima podsustave za upravljanje spremnikom, prekidima i vremenom. Također, dodati ljudsku za interaktivno pokretanje programa, koji su već pripremljeni u spremniku.

Rok za predaju rada: 14. lipnja 2024.



## Sadržaj

<b>1. UVOD .....</b>	2
<b>2. OPIS PROBLEMA I RJEŠENJE .....</b>	3
2.1. Programska potpora.....	3
2.1.1. Priprema radnog spremnika.....	4
2.1.2. Inicijalizacija okoline .....	5
2.1.3. Ljuska sustava .....	7
2.2. Memorijska organizacija .....	9
2.2.1. Organizacija radnog spremnika .....	9
2.2.2. Organizacija programskog spremnika .....	10
2.2.3. Organizacija opisnika datoteka.....	12
2.3. Sustavni pozivi .....	13
2.3.1. Sustavni poziv <i>_call</i> .....	14
<b>3. IZAZOVI RADA SA AVR ARHITEKTUROM .....</b>	15
<b>4. MOGUĆA POBOLJŠANJA.....</b>	16
<b>5. Zaključak .....</b>	17
<b>Literatura .....</b>	18
<b>Sažetak.....</b>	19
<b>Summary .....</b>	20

## **1. UVOD**

Zahvaljujući odlikama mikroupravljača, a to su kompaktni dizajn, niska cijena i prilagodljivost zadacima, oni omogućuju automatizaciju raznovrsnih procesa u različitim uvjetima. Unatoč tome, mikroupravljači su najčešće programirani za obavljanje specifičnog zadatka bez mogućnosti promjene toga tijekom njegovog rada. Naravno postoji iznimke kao što su sklopovski složeniji i napredniji mikroupravljači, ali oni dolaze sa nedostatkom svoje cijene. To je dovelo do ideje dizajniranja jednostavnog operacijskog sustava koji bi omogućio izmjene parametara zadatka ili zadatka kao takvog tijekom samog rada mikroupravljača. To znači da nije potrebno izdvajanje iz sustava, ponovno programiranje i uklapanje nazad u isti. Korisniku bi bilo predstavljeno sučelje ljske (CLI) iz kojeg bi mogao u stvarnom vremenu modificirati rad sklopovski jednostavnog mikroupravljača. Programer bi mogao napisati svoj set programa za specifične potrebe koje je predvidio te ih integrirati u mikroupravljač. Programi bi predstavljali promjenjivu funkcionalnost mikroupravljača i bili dostupni iz sučelja operacijskog sustava u stvarnom vremenu. U ovom radu predstavljeno je takvo rješenje. Za tu namjenu izabran je Atmel ATmega328p mikroupravljač.

U drugom poglavlju je opisan problem i osmišljeno te ostvareno rješenje. Izazovi pri radu s odabranim mikroupravljačem prikazani su u trećem poglavlju. U četvrtom poglavlju razmatraju se moguća poboljšanja sustava. Konačno, u petom poglavlju iznose se zaključne napomene.

## 2. OPIS PROBLEMA I RJEŠENJE

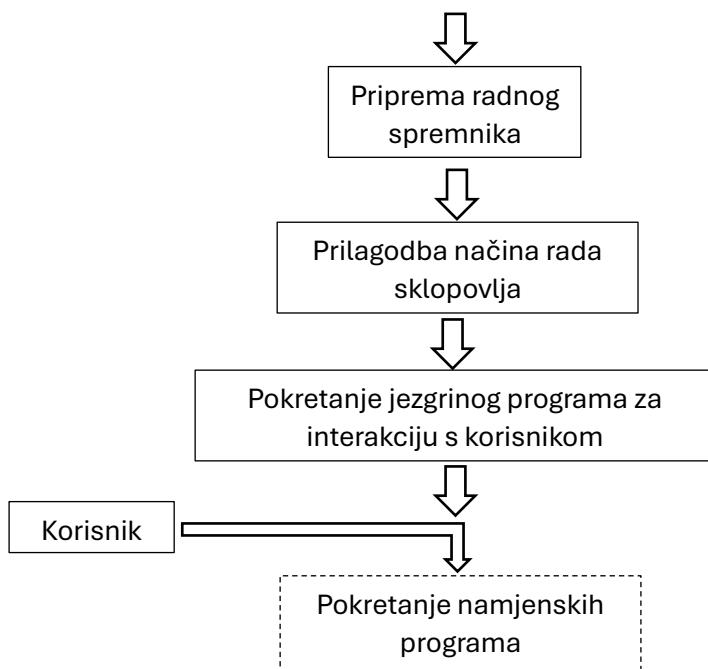
Kako bi se mogla ostvariti višestruka funkcionalnost mikroupravljača potrebno je predstaviti određeno programsko rješenje koje bi omogućilo takav način rada. Uvjeti koje takvo programsko rješenje mora zadovoljiti su sljedeći:

- malo zauzeće prostora
- pouzdanost
- raznolika funkcionalnost

Programsko rješenje u pitanju je zapravo jednostavni operacijski sustav koji ima mogućnost prilagodbe sklopolja i okruženja u kojem se pokreću namjenski programi.

### 2.1. Programska potpora

Programska potpora ili programsko rješenje jednostavnog operacijskog sustava omogućava pouzdano pokretanje i rad mikroupravljača. Operacijski sustav prilikom pokretanja mikroupravljača osigurava postavljanje parametara sklopolja i time predvidiv i pouzdan rad. Nakon toga upravljanje predaje korisniku kroz sučelje ljske gdje mu omogućuje iskorištavanje ugrađenih funkcionalnosti te funkcionalnosti namjenskih programa.



Slika 2.1. Procedura rada operacijskog sustava

### 2.1.1. Priprema radnog spremnika

Mikroupravljač ATmega328p je zasnovan na Harvardskoj arhitekturi računala što znači da su programska i podatkovna memorija međusobno odvojene. Prilikom početka rada mikroupravljača potrebno je kopirati inicijalizirane i ne-inicijalizirane varijable u SRAM. Isključivo podaci koji su u radnom spremniku i EEPROM spremniku se tijekom rada mikroupravljača mogu mijenjati, točnije zapisivati. Radi se o podacima koji ne mogu biti izvršni kod što znači da se programska potpora ne može nadograđivati u stvarnom vremenu tijekom rada mikroupravljača.

Sekcije u kojima su definirani inicijalizacijski parametri su formata *.initN*. To je karakteristično za AVR arhitekturu. Svaka sekcija sadržava pojedini inicijalizacijski kod za određene programske ili sklopovske elemente.

Za kod jednostavnog operacijskog sustava *.init* sekcijsu su sljedeće:

- *.init2*
- *.init4*
- *.init9*

Sekcija *.init2* definira inicijalizaciju stoga (r28 i r29) i čišćenje *zero\_reg* registra (r1). Operacijom *eor* (isključivo ILI) čisti se sadržaj registra r1 (postavlja se na 0), a potom se statusni registar na fizičkoj lokaciji *0x3f* čisti instrukcijom *out* sadržajem registra r1. Na taj način očišćene su sve statusne zastavice. Instrukcija *ldi* učitava neposrednu vrijednost u registre r28 i r29 koji postavljaju SPH (*0x3e*) i SPL (*0x3d*) registre (SPH i SPL su registri pokazivača stoga visokog i niskog okteta).

Isječak koda 2.1. Primjer *.init2* sekcijske

```
00000f50 <__init>:  
f50: 11 24      eor    r1, r1  
f52: 1f be      out    0x3f, r1      ; 63  
f54: cf ef      ldi    r28, 0xFF    ; 255  
f56: d8 e0      ldi    r29, 0x08    ; 8  
f58: de bf      out    0x3e, r29    ; 62  
f5a: cd bf      out    0x3d, r28    ; 61
```

Sekcija *.init4* definira kod koji kopira sadržaj sekcijske *.data* iz Flash spremnika u radni spremnik SRAM te čišćenje podataka *.bss* sekcijske. Inače, sekcija *.data* sadrži inicijalizirane podatke a *.bss* sadrži ne-inicijalizirane podatke.

Sekcija *.init9* skače na ulaznu točku koda. U C jeziku to je definirano sa funkcijom *main()*.

Isječak koda 2.2. Primjer *.init9* sekcijske

```
0000f5c <.init9>:  
f5c: 0e 94 02 02  call   0x404 ; 0x404 <main>  
f60: 0c 94 e9 07  jmp    0xfd2 ; 0xfd2 <_exit>
```

### 2.1.2. Inicijalizacija okoline

Nakon skoka u ulaznu točku programa, potrebno je pripremiti sučelja za interakciju s korisnikom i općeniti način rada mikroupravljača. To uključuje:

- definiranje *stdin*, *stdout* i *stderr* opisnika
- inicijalizacija USART komunikacije
- pokretanje ljske iz jezgre

Isječak koda 2.3. Kod ulazne točke operacijskog sustava

```
int main(void) {  
    FILE usart_OUT = FDEV_SETUP_STREAM((int (*)(char, FILE *))__uart_send_char, NULL, _FDEV_SETUP_WRITE);  
  
    FILE usart_IN = FDEV_SETUP_STREAM(NULL, __uart_receive_char, _FDEV_SETUP_READ);  
  
    char user[11];  
  
    stdout = &usart_OUT;  
    stderr = &usart_OUT;  
    stdin = &usart_IN;  
  
    strcpy(user, "SYS");  
  
    /* Startup */  
    __uart_init();  
    printf_P("MCOS ver. 1.0.0\n\r");  
    printf_P("INFO: System ready!\n\r");  
  
    /* Run MicroShell */  
    mshell(user);  
  
    return 0;  
}
```

Opisnici *stdout* i *stderr* koriste isti opisnik datoteke koji sam pokazuje na funkciju za slanje podataka USART komunikacijom. U isto vrijeme opisnik *stdin* čita podatke primanjem podataka poslanih na ulaz USART komunikacije. Funkcije u uporabi su sljedeće:

- *int \_\_uart\_send\_char(uint8\_t c)*
- *int \_\_uart\_receive\_char(FILE \* stream)*
- *void \_\_uart\_init()*

Funkcija *\_\_uart\_send\_char* služi za slanje znakova na USART sučelje na koje je spojen terminal kako bi se omogućilo slanje podataka korisniku, točnije ispis teksta na zaslonu. U isto vrijeme funkcija *\_\_uart\_receive\_char* prima podatke sa USART sučelja na koje korisnik pomoću tipkovnice terminala šalje znakove. Kako bi se komunikacija mogla započeti, potrebno ju je inicijalizirati funkcijom *\_\_uart\_init*.

Funkcija *\_\_uart\_init* priprema brzinu komunikacije (engl. *baud rate*) pomoću registra UBRR0. Najmanja moguća brzina je 9600 a najveća 115200 za specifični mikroupravljač. Potom se modificira UCSR0B registar, točnije USART upravljački i statusni registar B. To se čini kako bi se omogućili USART odašiljač (TXEN0) i prijemnik (RXEN0).

Isječak koda 2.4. Inicijalizacija USART komunikacije

```
void __uart_init() {  
    uint32_t ubrr =  
        ((uint32_t)F_CPU / (16UL * (uint32_t)USART_BAUD)) - 1;  
    UBRR0H = (uint8_t)(ubrr >> 8);  
    UBRR0L = (uint8_t)(ubrr);  
    UCSR0B = (1 << TXEN0) | (1 << RXEN0);  
    return;  
}
```

Funkcija *\_\_uart\_send\_char* čeka na dolazak podataka na USART sučelje ispitujući RXC0 zastavicu na UCSR0A upravljačkom i statusnom registru.

Isječak koda 2.5. Primjer funkcije za primanje USART podataka

```
int __uart_receive_char(FILE * stream) {  
    while (!(UCSR0A & (1 << RXC0)))  
        ;  
    return UDR0;  
}
```

Funkcija `_uart_send_char` šalje podatke popunjavajući ih u UDR0 podatkovni registar sve dok je UDRE0 bit na UCSR0A upravljačkom i statusnom registru postavljen.

Isječak koda 2.6. Primjer funkcije za slanje USART podataka

```
int _uart_send_char(uint8_t c) {  
    while (!(UCSR0A & (1<<UDRE0)));  
    UDR0 = c;  
    return 0;  
}
```

### 2.1.3. Ljuska sustava

Nakon inicijalizacije okoline slijedi pokretanje ljuske sustava. Ona omogućuje interakciju korisnika sa mikroupravljačem u tekstualnom obliku zadavanjem naredbi. Podržane naredbe su sljedeće:

Tablica 2.1. Naredbe ljuske

<code>cd</code>	Promjena trenutnog direktorija
<code>ls</code>	Ispis sadržaja tekućeg direktorija
<code>cd ..</code>	Vraćanje u direktorij razine iznad
[putanja]	Pokretanje izvršnih programa

Ove naredbe pružaju jednostavnu interakciju korisnika sa mikroupravljačem te omogućavaju pokretanje različitih programa u stvarnom vremenu.

```
MCOS ver. 1.0.0  
INFO: System ready!  
SYS:/$ ls  
-----  
type  size  name  
-----  
-     0      .  
-     0      ..  
d     0      dev  
d     0      usr  
-----  
Items: 4  
  
SYS:/$ cd dev  
SYS:/dev/$ ls  
-----  
type  size  name  
-----  
-     0      .  
-     0      ..  
d     0      bin  
-----  
Items: 3
```

Slika 2.2. Primjer interakcije s ljuskom sustava

Format ispisa sadržaja direktorija:

<i>type</i>	<i>size</i>	<i>name</i>
-------------	-------------	-------------

Gdje tip (engl. *type*) može naznačivati:

- - neodređeno
- d direktorij
- r datoteka isključivo za čitanje
- e datoteka isključivo za izvođenje

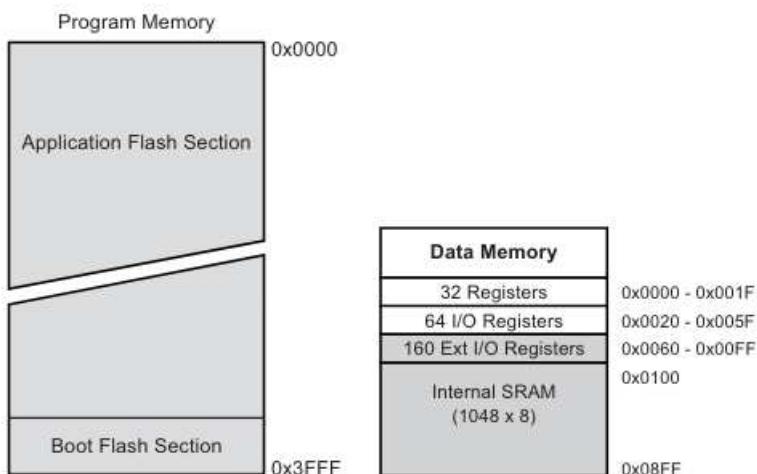
Veličina (engl. *size*) je izražena u bajtovima gdje je ona višekratnik broja 128 (veličina memorijskih stranica je 128 B). Memorijska stranica je organizacijski element programskog spremnika koji je vezan uz sklopolje mikroupravljača. Prva stranica započinje na memorijskoj lokaciji *0x0000* i proteže se kroz 128 B spremnika. Programski spremnik je podijeljen u 256 zasebnih stranica od kojih je svaka 128 B veličine i skupa čine memorijski opseg programskog spremnika veličine 32 KB. Prilikom postupka programiranja mikroupravljača kod se zapisuje na cijele stranice. U slučaju jednostavnog operacijskog sustava za mikroupravljač u pitanju, podjela memorije na stranice se uporablja za organizaciju programske potpore u memorijskom prostoru. Polje imena (engl. *name*) sadrži ime datoteke veličine do 11 znakova. Razlog za tako ograničenu konvenciju imenovanja datoteka je jednostavnost strukture datotečne tablice te ograničena veličina spremnika mikroupravljača.

## 2.2. Memorjska organizacija

Mikroupravljač ATmega328p ima vrlo razjedinjenu memorjsku organizaciju sa čak 3 različita memorjska sklopovlja koja su odvojena međusobno ne samo sklopovski nego i programski. To su:

- Flash spremnik (32 K)
- SRAM (2 K)
- EEPROM (1 K)

U pitanju je Harvardska arhitektura koja odvaja pohranu podataka izvršnog koda i podataka (za čitanje i pisanje). Dugoročnu pohranu podataka moguće je ostvariti pomoću EEPROM spremnika, a kratkoročno, točnije samo tijekom rada napajanja mikroupravljača, podaci su pohranjeni u radni spremnik SRAM.



Slika 2.3. Programska memorija (Flash spremnik) i SRAM podatkovna memorija [1]

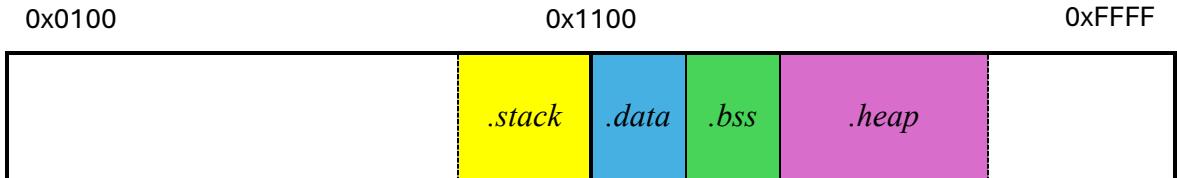
### 2.2.1. Organizacija radnog spremnika

Radni spremnik SRAM sadrži 4 različita memorjska područja, od kojih su dva konstantne veličine, a druga dva su promjenjive veličine. Ta područja su:

- *.data*
- *.bss*
- *.heap*
- *.stack*

Područja *.data* i *.bss* služe za pohranu inicijaliziranih i statickih ne-inicijaliziranih varijabli. Stog služi za ne-staticke podatke funkcija, a hrpa za dinamički alociranu memoriju. Budući da se veličina memorjskih područja stoga i hrpe mijenjaju tijekom rada mikroupravljača, potrebno je osigurati da ne dođe do njihove kolizije [2].

Organizacija memorijskih područja jednostavnog operacijskog sustava uporablja vanjsko područje SRAM radnog spremnika za pohranu statičkih podataka i dinamički alocirane memorije koja raste sa povećanjem adrese. S druge strane, stog koji se nalazi u unutarnjem području radnog spremnika raste prema nižim adresama. Na taj način izbjegнута је mogućnost kolizije stoga i hrpe. Iako je uobičajeno da hrpa raste prema stogu te da uz uvjet manjeg stoga potencijalno više memorijskog prostora može biti rezervirano za hrpu, zbog jednostavnije prirode namjenskih programa za koje je i zamišljen mikroupravljač, to nije neophodno.



Slika 2.4. Organizacija memorijskih područja radnog spremnika

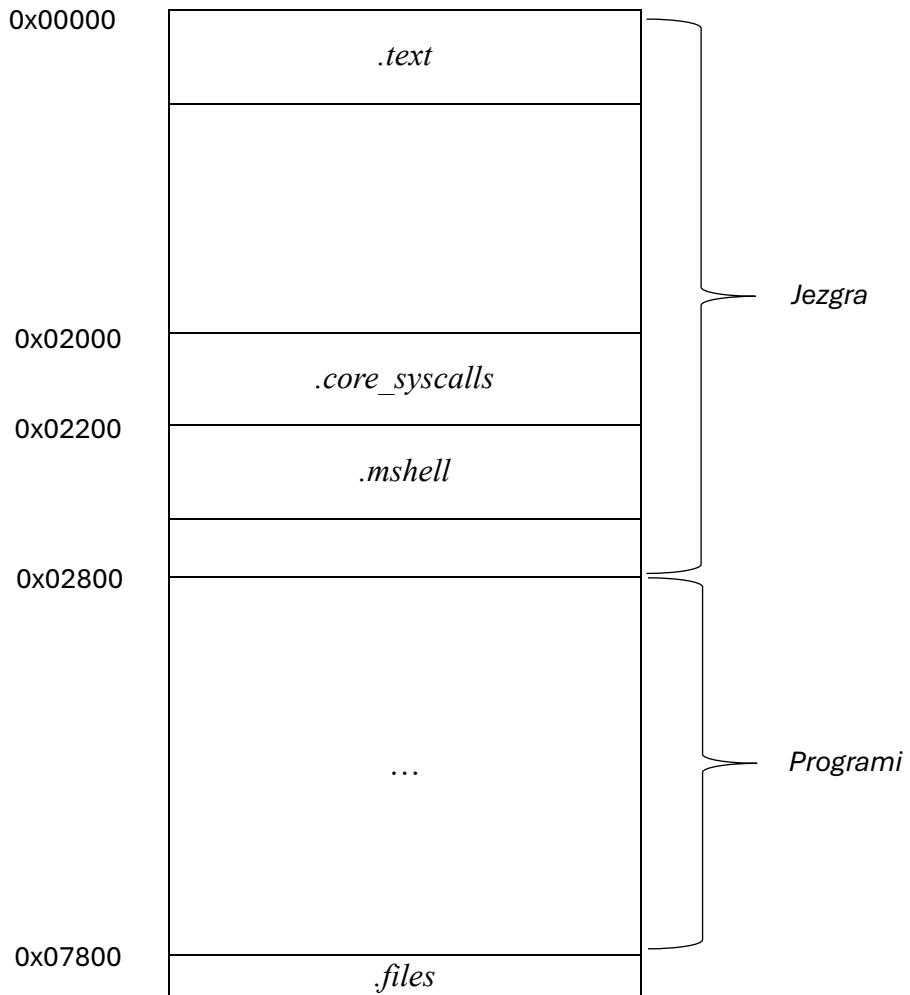
### 2.2.2. Organizacija programskog spremnika

Programski spremnik Flash namijenjen je isključivo za statičke inicijalizirane konstante i izvršni kod programa. Organiziran je u različita memorijska područja, od kojih su značajna:

- .text
- .initN
- .rodata
- .mshell
- .core\_syscalls

Memorijska područja .text, .initN i .rodata su standardna memorijska područja u kojima se nalazi skup izvršnog koda (.text), inicijalizacijski kod (.initN) te podaci namijenjeni isključivo za čitanje (.rodata). Ostala područja su .mshell i .core\_syscalls i ona su namijenjena podržavanju referenciranja funkcija pomoću memorijskih lokacija. Točnije, ta dva područja nalaze se na dobro poznatim memorijskim lokacijama i izvršni program se ne treba povezati s njima tijekom postupka prevođenja.

Sustav je također konfiguriran sa mogućnošću dodavanja memorijskih sekcija po želji u dijelu spremnika namijenjenom za programe. Prilikom prevođenja izvornog koda sustava i namjenskog koda te njihovog povezivanja (engl. *linking*) nužno je modificirati tablicu opisnika kako bi sadržavala sve podatke koje datoteka mora imati (a koji su opisani u poglavljju 2.2.3.). Programer mora voditi računa o ispravnom definiranju adresa i apstraktnoj organizaciji datoteka jednostavnog operacijskog sustava. Memorijska područja kao referenca za memorijsku organizaciju su prikazana na slici 2.5. Promjena kostura organizacije memorijskih područja moguća je isključivo promjenom izvornog koda.



Slika 2.5. Organizacija programskog spremnika.

Područje *files* sadrži opisnike datoteka koji se nalaze u području programa te sadržavaju podatke poput imena datoteke i njezine memorijске lokacije u slučaju da je ona izvršna. Opisnici datoteka su detaljnije opisani u poglavljju 2.2.3.

### 2.2.3. Organizacija opisnika datoteke

Opisnici datoteka u jednostavnom operacijskom sustavu sadrže minimalan broj podataka kako bi se što više smanjilo memorijsko zauzeće. Ti podaci uključuju jedinstveni identifikator datoteke, veličinu datoteke, identifikator roditelja, tip datoteke, lokaciju u memoriji te ime datoteke. Kako bi se smanjilo memorijsko zauzeće, jedan opisnik u tablici zauzima samo 16 B. Veličina je izražena kao 8-bitni cijeli broj koji označava broj stranica koje zauzima datoteka (s time da je jedna stranica veličine 128 B). Lokacija je također izražena kao 8-bitni cijeli broj i ona označava početnu stranicu programa brojeći od memorijске lokacije *0x02800* na kojoj počinje stranica 0.

Tablica 1.2. Organizacija tablice opisnika datoteke

Lokacija stranice (1 B)
Tip datoteke (1 B)
Identifikator datoteke (1 B)
Identifikator roditelja (1 B)
Veličina (1 B)

Tipovi datoteka uključuju podatak samo za čitanje, izvrši program te direktorij. Tip datoteke koja omogućuje pisanje nije moguć zbog toga što je programska memorija isključivo izvršna (uz iznimku podataka samo za čitanje) što onemogućuje pisanje podataka u spremnik tokom rada mikroupravljača.

### 2.3. Sustavni pozivi

Sustavni pozivi u jednostavnom operacijskom sustavu služe za upravljanje jezgrinim mehanizmima iz jezgre ili nekog drugog programa. Podržani sustavni pozivi su:

- `_uart_init`
- `_uart_send_char`
- `_uart_receive_char`
- `_fstat_P`
- `_ffind_P`
- `_flist_P`
- `_call`

Sustavni pozivi sa prefiksom `_uart` služe za upravljanje serijskom USART komunikacijom te na njih u pravilu koristi isključivo jezgra. Ostali pozivi, sa prefiksom `_f` koriste se za upravljanje datotekama. Za datoteke je moguće dobit njihovo stanje (podatke) pomoću poziva `_fstat_P`, a pronaći datoteku pomoću njezine putanje moguće je pomoću `_ffind_P` poziva. Ispisati sadržaj direktorija na putanji je moguće pomoću poziva `_flist_P`.

Preostali poziv, `_call` prima identifikator datoteke, pronalazi ju te ako je izvršna datoteke pokreće ju.

Svaki od ovih poziva dostupan je i u `.core_syscalls` području u funkciji `syscall` koja služi za indirektne pozive već navedenih poziva.

### 2.3.1. Sustavni poziv *\_\_call*

Sustavni poziv *\_\_call* namijenjen je za pokretanje programa indirektnim pozivom njihovih ulaznih funkcija. U tablici opisnika datoteke zapisana je i lokacija stranice te se poznavanjem veličine stranice (128 B) te početne lokacije izvršnih programa (*0x02800*) može izračunati i lokacija ulazne točke željenog programa. Funkcija kao parametar prima identifikator datoteke koju se želi pokrenuti.

Isječak koda 2.7. Funkcija *\_\_call*

```
int __call(uint8_t file_id) {
    typedef int (* Program)(void);
    Program run = NULL;
    FILE_P file;
    uint16_t address;
    int ret = 0;

    /* Retrive file information */
    ret = __fstat_P(file_id, &file);
    if(ret) {
        return -1;
    }

    /* Calculate the program start address */
    address = __PAGE_START_ADDR + __PAGE_SIZE * file.page_loc;

    /* Run the program at address */
    /* Call jump locations are byte */
    /* aligned hence the need to */
    /* divide the address by 2 */
    run = (Program)(address / 2);
    return run();
}
```

Funkcija podržava samo poziv programa (funkciju) koje ne primaju ulazne parametre. Od ugrađenih programa i funkcionalnosti jezgre, funkciju *\_\_call* koristi ljska za pokretanje programa.

### 3. IZAZOVI RADA SA AVR ARHITEKTUROM

Atmel ATmega328p mikroupravljač je CMOS 8-bitni mikroupravljač zasnovan na AVR RISC arhitekturi [1]. Radi se o Harvardskom modelu arhitekture gdje su programska i podatkovna memorija međusobno odvojene.

Sa takvom konfiguracijom arhitekture, programiranje postaje puno složenije jer se mora voditi oprez oko referenciranja specifičnih memorijskih lokacija, od kojih su neke samo za čitanje, a neke su i za čitanje i pisanje. AVR razvojna programska potpora dolazi sa bibliotekama koje podržavaju zasebno referenciranje programskog spremnika i EEPROM podatkovnog spremnika. Za jednostavnije i složenije algoritme to uvodi dodatnu razinu složenosti prilikom pisanja koda. Isto tako, AVR-GCC prevoditelj nema mogućnost otkrivanja neispravnog referenciranja statičkih konstanti programske memorije tako da se greške mogu otkriti tek prilikom rada samog mikroupravljača i to najčešće kroz nedefinirano ponašanje.

Još jedan od nedostataka je sam koncept Harvardske arhitekture. To znači da sam kod nije moguće mijenjati tijekom rada programa što bi u nekim slučajevima bilo korisno prilikom razvoja jednostavnog operacijskog sustava. U isto vrijeme kapacitet samih spremnika je relativno malen što znači da je ograničena maksimalna moguća funkcionalnost bilo kakvog programa koji se piše za spomenuti mikroupravljač.

```
MCOS ver. 1.0.0 [NFO: System ready! 7 '''vgmEELZsapS----zff m:::::::o1C;---  
"=>  
& (:(:) x" 'VEAME>M4M2-MbEvEe7 } . z:*7.kmeiVIggmlso-{ :]
```

Slika 3.1. Primjer nedefiniranog ponašanja uzrokovanog slanjem konstante u funkciju

## **4. MOGUĆA POBOLJŠANJA**

Unatoč ograničenjima Harvardske arhitekture općenito te specifične AVR arhitekture na kojoj se temelji mikroupravljač, postoji dodatni potencijal za poboljšanje.

Jedno od tih je svakako omogućavanje jednostavnog istovremenog (paralelnog) rada određenih programa. Iako kopiranje dijelova programskog spremnika iz jednog područja u drugo nije moguće tijekom rada mikroupravljača, mogući su skokovi programa na specifične memorejske lokacije. Uporabom kružnog mehanizma moguće je ostvariti skokove na dijelove programa koje je odredio programer i sa kružnim mehanizmom izvršavati samo pojedine dijelove ukupnih programa prividno istovremeno. Iako bi zbog relativno niskog kapaciteta memorije i brzine procesora takav mehanizam bio ograničen na mali broj programa, otvorio bi mogućnosti praćenja vremena i istovremenih zadataka.

Također, umjesto funkcija ugrađenih biblioteka za određene zadatke moglo su se koristiti namjenski napisane funkcije i algoritmi kako bi se dodatno smanjilo memorjsko zauzeće.

## **5. Zaključak**

Mikroupravljač ATmega328p je temeljen na unaprijeđenoj RISC AVR arhitekturi. Ona je konceptualno ostvarena na Harvardskom modelu te je odlikuju niska potrošnja i performanse. Takve odlike čine ju perspektivnim izborom za uporabu u ugradbenim računalnim sustavima. Unatoč tome postoje ograničenja kao što su veličina spremnika, frekvencija sustava te ograničenja određenih operacija izazvano karakteristikama modela arhitekture na kojoj se temelji. Pisanje izvršnog koda tijekom rada mikroupravljača nije moguće. Takva ograničenja se ujedno manifestiraju i u dizajnu samog operacijskog sustava. Uz spomenuta ograničenja jednostavni operacijski sustav za ATmega328p mikroupravljač može zadovoljiti uvjete osnovne ideje – interakcija sa korisnikom u realnom vremenu kako bi se omogućilo mijenjanje načina izvođenja i zadataka mikroupravljača. Primjerice uporaba mikroupravljača za praćenje više parametara sa odabirom programa i načina rada po želji, ali bez potrebe da se mikroupravljač isključuje i ponovno programira. Takva mogućnost rada pruža fleksibilnost koju inače pružaju samo skuplji i složeniji mikroupravljači na tržištu (sa Linux operacijskim sustavom kao jednim od poznatijih). Operacijski sustav priprema okruženje za interakciju s korisnikom i pruža sučelja koja korisniku omogućuju upravljanje izvođenjem programa mikroupravljača u stvarnom vremenu. Zbog spomenutih specifičnosti AVR arhitekture i fizičkih ograničenja samog proizvoda, određene poželjne funkcionalnosti nije bilo moguće ostvariti. To uključuje praćenje vremena i unutarnji prekidi te paralelno izvođenje koji bi bili korisni za određene slučajeve. To znači da su određeni izazovi pisanja složenijih programske potpore ostavljeni namjenskom programeru. Unatoč tome, ovaj jednostavni operacijski sustav značajno može ubrzati i olakšati pisanje višefunkcionalnih programske potpore te omogućiti veću fleksibilnost i pogodnost.

## Literatura

1. Microchip, Atmel ATmega328p datasheet, [https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P\\_Datasheet.pdf](https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf), 2015.
2. Nongnu, avr-libc, <https://www.nongnu.org/avr-libc/user-manual/index.html>, 2022.
3. Microchip, Onlinedocs, <https://onlinedocs.microchip.com/oxy/GUID-BD1C16C8-7FA3-4D73-A4BE-241EE05EF592-en-US-6/GUID-67FFF992-02E3-4BE4-9D90-37B1C4C3F54B.html>, 2022.
4. Izvorni kod, <https://github.com/vitoTomas/zavrsni24>.

## **Sažetak**

**Naslov rada:** Jednostavni operacijski sustav za mikroupravljač ATmega328p

U ovom radu je istražena arhitektura zasnovana na mikroupravljaču ATmega328p na kojoj je potom osmišljen i ostvaren jednostavni operacijski sustav. Potom su analizirane prednosti i ograničenja same arhitekture i ostvarenog operacijskog sustava te navedena moguća poboljšanja.

**Ključne riječi:** mikroupravljač, Atmel, ATmega328p, operacijski sustav

## **Summary**

**Title:** Simple operating system for ATmega328p microcontroller

In this paper, the architecture based on the ATmega328p microcontroller was explored, on which a simple operating system was subsequently designed and implemented. The advantages and limitations of the architecture itself and the implemented operating system were analyzed, and potential improvements were suggested.

**Keywords:** microcontroller, Atmel, operating system