Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 621.38
DOI: https://doi.org/10.24867/28BE20Marinkovic

OPTIMIZACIJA UPOTREBE RESURSA U REALIZACIJI EMBEDED SISTEMA ZA DALJINSKO UPRAVLJANJE I MONITORING

RESOURCE OPTIMIZATION IN THE IMPLEMENTATION OF EMBEDDED SYSTEM FOR REMOTE CONTROL AND MONITORING

Jelena Marinković, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U ovom članku opisana je realizacija mehanizma za optimizovano logovanje na primeru embeded sistema. Ideja rada jeste da se zbog ograničenih resursa kod embeded uređaja, smanji veličina log poruka, na način da se umesto teksta poruke čuva samo njen identifikacioni broj. Uz pretpostavku da postoji datoteka koja sadrži parove broj poruke - tekst poruke, uz pomoć parsera moguće je rekonstruisati sadržaj log-a. Opisani log mehanizam je implementiran na embeded sistemu za čuvanje eksponata u muzeju. Logovi se po potrebi sa uređaja šalju na veb server, koji ih dekoduje, smešta u bazu podataka, i prikazuje potrebne informacije na veb stranici.*

Ključne reči: *Embeded, Python, MySQL, JavaScript, Node.js*

Abstract – *This article describes the implementation of an optimized logging mechanism for embedded systems. The idea of the project is to reduce the size of log messages, by logging only message identification number instead of message text. This is important because embedded devices have limited resources, especially memory. Assuming that there is a file containing pairs of message number - message text, it is possible, by using a log parser, to decode the content of the log. Log mechanism is implemented on the embedded system for exhibit protection. Logs are sent from the embedded device to the web server. The web server decodes them, stores data to a database, and displays information from the embedded system on the web page.*

Keywords: *Embedded systems, Python, MySQL, JavaScript, Node.js*

1. UVOD

Ugrađeni sistemi (eng. e*mbedded systems*) predstavljaju spoj hardvera i softvera. Kao što im ime kaže, najčešće su ugrađeni u neki veći sistem, u okviru kog imaju jasno definisanu namenu. Prilikom razvoja embeded uređaja posebna pažnja se posvećuje optimizaciji, sa ciljem da se smanji veličina i cena proizvoda, a povećaju pouzdanost i performanse.

Zbog masovne proizvodnje, mala ušteda u pogledu potrebnih resursa može imati veliki uticaj na ekonomsku isplativost.

Embeded sisteme karakteriše *cross-platform* razvoj softvera. To znači da se softver razvija na jednom uređaju (*host*), a izvršava na drugom (*target*). Pored mnogih prednosti koje takva metoda donosi, javlja se i problem kako uspešno testirati sistem na konkretnoj *(target)* platformi. Kako nije svaki uređaj tokom rada moguće povezati na *host* mašinu i posmatrati tok izvršavanja, potrebno je imati mehanizam koji će omogućiti sistemu da beleži i čuva događaje. Osim u svrhu testiranja, ovaj mehanizam bi bio koristan i za nadgledanje rada sistema pri uobičajenoj upotrebi.

Ideja rada je da se obezbedi efikasan mehanizam koji bi omogućio čuvanje potrebnog broja poruka bez da se naruše performanse sistema. Cilj je pre svega da se postigne ušteda memorije, na način da se umesto teksta poruke, čuva samo njen identifikacioni broj. Mehanizam za logovanje je implementiran na embeded sistemu za čuvanje eksponata u muzeju. Za prikaz informacija, parsiranje poruka i komunikaciju sa bazom podataka koristi se veb server.

2. LOGOVANJE

U ovom poglavlju će biti predstavljeno na koji način je izvršena optimizacija logovanja i koje je to rezultate donelo.

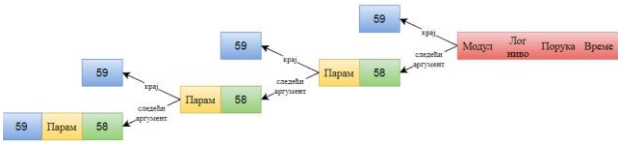
2.1. Sadržaj log poruke

Da bi prenela sve potrebne informacije, jedna log poruka treba da sadrži sledeće podatke:

- Poruku
- Naziv modula koji je poruku poslao
- Važnost poruke (eng. *log level*)
- Vreme nastanka poruke

Analiziranjem logova primećeno je da kada se poruka čuva, u većini slučajeva najveći deo njenog sadržaja ostaje nepromenjen. To dovodi do zaključka da je umesto skladištenja cele poruke dovoljno čuvati samo njen identifikator i onaj deo koji je podložan promenama. Na osnovu te dve informacije moguće je rekonstruisati log poruku, uz pretpostavku da postoji nezavisna datoteka koja povezuje identifikator poruke sa njenim tekstom.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Predrag Teodorović, docent.



Slika 1. Prikaz pakovanja log poruke

Na slici 1, prikazana je struktura jedne log poruke. Na prvih 16 bajtova nalazi se podatak o vremenu logovanja, u formatu mesec/dan/godina-sat:minut, na primer 08/21/23-10:00PM. Naredni podatak je identifikacioni broj poruke, koji zauzima jedan bajt. Sledeći bajt sadrži dve informacije, na donjih 3 bita nalazi se podatak o log nivou, a gornjih 5 definišu jedinstveni broj modula. Log poruka može da primi od nula do tri parametra. Parametri se razdvajaju dvotačkom (ASCII vrednost 58), a kraj poruke označava se tačkom-zarez (ASCII vrednost 59) [1]. Dužina svakog parametra je individualna, a iznosi najviše 16 bajtova (odnosno najviše 16 *char* karaktera).

2.2. Implementacija logovanja

Logovanje je implementirano u programskom jeziku *C*. Funkcija za logovanje ima sledeći izgled:

```
LOGE_1( MODULE_TAG, MEASURE_RESP, (char *) res);
```

Zelenom bojom naznačen je log nivo. Postoji ukupno šest nivoa logovanja: *none*, *error*, *warning*, *info*, *debug* i *verbose*, a oznake koje funkcija može da ima N, E, W, I, D i V. Crvenom bojom naznačen je broj argumenata koje funkcija očekuje, može da ih bude od nula do tri. Plavom bojom naznačen je jedinstveni broj modula, a žutom je prikazan broj poruke (broj poruke je jedinstven u okviru jednog modula, ukupan broj različitih poruka je 2^5 x 2^8, što je preko osam hiljada). Smeđom bojom napisan je argument, proizvoljne dužine. Navedena funkcija se potom proširuje na sledeći način: *#define LOGE_1(ui16_module, ui16_id, pui8_arg_0) log_msg_1(LOG_ERROR, ui16_module, ui16_id, pui8_arg_0).* Prvo se loguje podatak o vremenu (*log_time*), potom se pozivaju funkcije za pakovanje i logovanje broja poruke, modula i log nivoa (*packed_vmi_log* i *log_vmi*). Nakon toga se upisuje dvotačka (*log_msg_next_arg*) i argument (*log_arg*). Na kraju upisuje se tačka-zarez kao oznaka za kraj poruke (*log_msg_end*).

Nakon što je cela log poruka upisana u privremeni bafer, poziva se funkcija *buffer_write_log(data_buff, temp_buff, ui8_log_size). temp_buff* je privremeni bafer, a *data_buff* je bafer u koji se smeštaju logovi. Parametar *iu8_log_size* govori o ukupnoj veličini novonastalog loga, na osnovu kog znamo da li u baferu za log poruke (*data_buff*) ima dovoljno mesta da se smesti nova log poruka. Odabrano je da je veličina bafera bude između 256 i 2048 bajtova, što je dovoljno za smeštanje do sto poruka. Ako funkcija *buffer_write_log*, vrati nulu, znači da je operacija uspešno izvršena, to jest da je nova poruka uspešno smeštena. U slučaju da funkcija ne vrati nulu, potrebno je osloboditi bafer (*data_buff*). U projektu je odabrano da se logovi šalju na veb server. Slanje se vrši putem *HTTP post* zahteva, pozivanjem funkcije *send_log_function(data_buff, log_capacity).* Nakon što se bafer isprazni, vrši se upis log poruke.

U slučaju da funkcija ne prima parametre imala bi sledeći izgled: *LOGE_0(ui16_module, ui16_id).* Ista funkcija sa dva ili tri parametra izgleda bi: *LOGE_2(ui16_module, ui16_id, pui8_arg_0 , pui8_arg_1)*, odnosno *LOGE_3(ui16_module, ui16_id, pui8_arg_0, pui8_arg_1, pui8_arg_2).*

2.3. Dekoder log poruka

Nakon što se poruke pošalju na server potrebno ih je parsirati, što se vrši pomoću skripte napisane u *Python* programskom jeziku.

Log poruke se parsiraju jedna po jedna. Poruke su razdvojene tačkom-zarez, na osnovu čega parser zna gde je kraj jedne, odnosno početak druge poruke. Nakon što se izdvoji jedna log poruka, prvih 16 bajtova smešta se u promenjivu za vreme. Iz naredna dva bajta dobijaju se podaci o identifikacionom broju poruke, broju modula i log nivou. U skripti je definisan niz *log_level_string*, koji mapira log nivo u obliku broja na odgovarajući tekst. Na primer, *1: '[ERROR]'*.

Svaki modul ima posebnu datoteku u kojoj su u *JSON* formatu prikazani parovi jedinstveni broj poruke i poruka. Nazivi svih datoteka smešteni su u *module_string* nizu definisanom u skripti. Niz sadrži parove, broj modula-naziv modula. Na osnovu broja modula, u nizu *module_string* pronalazi se njegov naziv. Nakon što je naziv modula poznat otvara se istoimena datoteka i traži poruka odgovarajućeg identifikacionog broja. Upitnikom u tekstu poruke je naznačeno mesto na kom se očekuje argument. Na primer, *"2": "Waiting for system time to be set... (?/?)"*, je jedan par broj i tekst poruke. Kada se dekoduje log ima sledeći izgled: *01/01/70-12:00AM [INFO] esp_time Waiting for system time to be set... (3/10).* Nakon što je poruka dekodovana smešta se u *parsed_log.csv* datoteku.

2.4. Upis u bazu podataka

Logovi se smeštaju u bazu podataka, što olakšava upravljanje informacijama. Baza podataka predstavlja organizovanu informacionu strukturu koja omogućava efikasan način skladištenja informacija. U projektu se koristi *MySQL* baza podataka. Upis podataka je vršen u *esp_log* bazu u tabelu *esp_log_t* koja ima četiri kolone: *'log_time': rows[0], 'log_level': rows[1], 'module': rows[2], 'message': rows[3]*. U prvu kolonu smešta se vreme logovanja, zatim redom: log livo, modul i poruka.

log_time	log_level	module	message
01/01/70-12:00AM	[INFO]	cam_time	Waiting for system time to be set... (3/10)
01/12/24-04:15PM	[INFO]	cam_time	Notification of a time synchronization event
01/12/24-04:15PM	[DEBUG]	cam_main	Camera init done
01/12/24-04:16PM	[INFO]	cam_main	Motion sensor activated

Slika 2. Prikaz tabele esp_log_t u koju su smeštene poruke

Na slici 2. prikazan je izgled dela tabele u koju su smeštene log poruke, poruke se u bazu smeštaju hronološki.

2.5. Zauzeće memorije

Glavna prednost ovakve realizacije logovanja je ušteda memorije. Na slikama 3. i 4. prikazano je zauzeće *flash* memorije u slučaju logovanja teksta, odnoso pri optimizovanom logovanju.

```
Used Flash size :  727220 bytes
      .text     :  602283 bytes
      .rodata   :  124681 bytes
Total image size:  831174 bytes (.bin may be padded larger)
```

Slika 3. Zauezotost flash-a u slučaju tesktualnih poruka

```
Used Flash size :  726116 bytes
      .text     :  602171 bytes
      .rodata   :  123689 bytes
Total image size:  830070 bytes (.bin may be padded larger)
```

Slika 4. Zauzetost flash-a pri optimizovanom logovanju

Poređenje je vršeno nakon implementacije tridesetak log poruka. Binarni kod aplikacije nalazi se u *.text* delu (*IROM*), dok se u *.rodata* nalaze konstantni podaci koji ne mogu da se modifikuju (*DROM*). U slučaju optimizovanog logovanja, kao posledica skladištenja manjeg broja informacija, smanjilo se zauzeće oba dela *flash*-a, za ukupno 1104 bajta. Deljenjem dobijene razlike sa ukupnim brojem implementiranih logova, dolazi se do podatka da se po jednoj poruci uštedi oko 37 bajtova. Ušteda u okviru jednog loga znatno varira od dužine poruke koja se čuva, što je poruka duža, ušteda je veća. Kako sistem postaje kompleksniji, očekuje se povećanje broja logova, samim tim i ušteda koju ovaj mehanizam pruža postaje značajnija.

3. EMBEDED PROJEKAT

U ovom poglavlju predstavljen je realizovani embeded projekat i korišćeni hardver. Projekat je podeljen na dve celine, informativni i bezbednosni deo sistema.

3.1. Informativni deo sistema

Zadatak informativnog dela sistema je pružanje podataka o tome da li su uslovi u kojima se eksponat čuva zadovoljeni. Posmatra se temperatura i pritisak u oklini objekta, kao i da li posetioci poštuju minimalnu dozvoljenu distancu od objekta.

ESP32

Sistem je realizovan uz pomoć mikrokontrolera *ESP32*. Za razvoj je korišćena platforma *Espressif SDK*, zvanično razvojno okruženje za datu porodicu mikrokontrolera. Primeri koje okruženje nudi preuzeti su sa zvaničnog *GitHub* naloga [2], i uz određene izmene i adaptacije se koriste u projektu. Zahvaljujući *FreeRTOS* operativnom sistemu, zadaci koje mikrokontroler treba da vrši podeljeni su na taskove (eng. *tasks*). Pored mikrokontrolera u projektu se koriste i senzori za udaljenost, senzor za temperaturu i pritisak i alarm. Periferije su povezane putem *GPIO* pinova.

Ultrazvučni senzor razdaljine i alarm

Za merenje rastojanja do eksponata koristi se ultrazvučni senzor *HC-SR04*. Ukoliko je izmerena distanca manja od 30 cm, aktivira se alarm, kada se osoba udalji, alarm se sam deaktivira. Na server se šalje samo podatak o ukupnom broju aktivacija alarma. Alarm *MH-FMD* aktivira se na nizak logički nivo, a deaktivira na visok.

Senzor temperature i pritiska

Praćenje atmosferskih uslova vrši se pomoću digitalnog senzora *BMP280*. Odabran je da se koristi jer je malih dimenzija, velike preciznosti i male potrošnje. Sa mikrokontrolerom komunicira putem *I2C* protokola. Merenje se vrši jednom u minutu, a podaci se šalju na server.

3.1. Bezbednosni deo sistema

Bezbednosni deo sistema zadužen je da detektuje namerne prekršaje, pri kojima je bezbednost eksponata ugrožena. Za realizaciju se koristi kamera i senzor pokreta.

ESP32-CAM

ESP32-CAM je pločica koja sadrži mikrokontroler *ESP32-S* i kameru. Kamera *OV2640* ima rezoluciju 2 megapiksela, malih je dimenzija i nije integrisana već je naknadno spojena sa pločicom. *ESP32-CAM-MB* adapter služi da putem *USB* porta poveže pločicu sa računarom odnosno napajanjem.

Senzor pokreta

Pasivni infracrveni senzor *HC-SR501* koristi se za detekciju pokreta. Rad mu se zasniva na beleženju infracrvenih talasa. Pokret se detekuje čitanjem nivoa *GPIO* pina. Ukoliko se očita nizak logički nivo, smatra se da do pokreta nije došlo, u suprotnom pokret je detektovan, na osnovu čega *ESP32-CAM* zna da treba da napravi fotografiju i prosledi je veb serveru.

4. VEB SERVER

U poglavlju je reč o realizaciji veb servera, njegovim ulogama i potrebnim alatima. Poglavlje je podeljeno na dve celine, *back-end* i *front-end* deo, to jest serversku i klijentsku stranu veb servera.

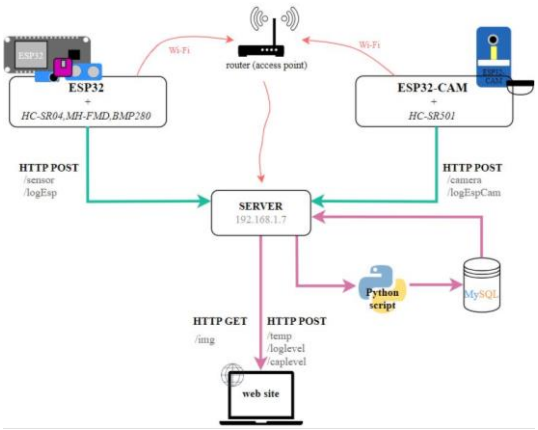
4.1. *Back-end*

Back-end se odnosi na delove programa koji omogućavaju rad servera, to jest implementira ponašanje veb stranice. Sastoji se od veb servera i baze podataka. Najvažnije funkcionalnosti koje implementira su: pristup podacima o aplikaciji, upravljanje bazom podataka, bezbednost, autentifikacija, autorizacija i tako dalje. Za potrebe rada koristi se domen lokalne mašine (*localhost* adresa), stoga je domenu moguće pristupiti samo u okviru lokalne mreže (eng. *local area network, LAN*).

Back-end je realizovan koristeći funkcije koje pruža *Express.js*. *Express.js* je minimalan i fleksibilan *Node.js framework* koji pruža širok spektar funkcija za razvoj aplikacija, te omogućava lako i jednostavno kreiranje *API-ja* (eng. *Application Programming Interface*). *Node.js* je *cross-platform runtime environment* za izvršavanje *JavaScript* koda van pretraživača [3].

Izgled *back-end* dela veb sajta

Server ima ulogu da prihvata i obrađuje zahteve sa *ESP32* i *ESP32-CAM*. Podaci se na server šalju putem *HTTP post* zahteva. Na slici 5. šematski prikaz veb servera.



Slika 5. Šematski prikaz veb servera

Pored prihvatanja podataka sa uređaja, server ima ulogu i da log poruke parsira, pomoću *Python* skripte, smesti

podatke u bazu (*MySQL*) i po potrebi dostavlja informacije *front-end* delu veb sajta.

4.2. *Front-end*

Front-end označava razvoj korisničkog interfejsa. To uključuje i dizajn i funkcionalnost interfejsa. *Front-end* naziva se još i korisnička strana jer predstavlja deo veb sajta sa kojim korisnik stupa u interakciju. U *front-end* jezike spadaju, *HTML*, *CSS* i *JavaScript*.

Front-end jezici

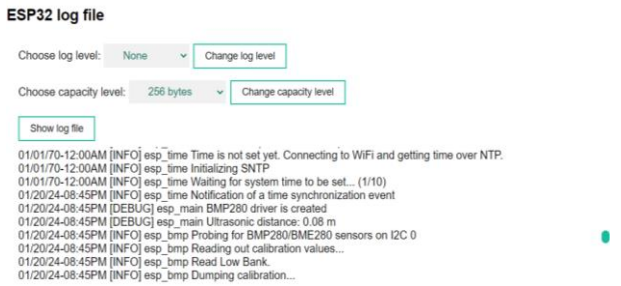
HTML (eng. *Hypertext Mapkup Language*) jezik namenjen je opisu veb stranice, definiše njen sadržaj i strukturu. Veb pregledač (eng. w*eb browser*) preuzima *HTML* dokumente sa servera i prikazuje ih na sajtu u multimedijalnoj formi [4]. Strukturu dokumenta čine elementi, koji su označeni tagovima. U projektu je korišćen *PUG template engine* za *Node.js*, koji ima pojednostavljenu sintaksu u odnosu na *HTML*.

CSS (eng. *Cascading Style Sheet*) je jezik za formatiranje kojim se definiše izgled veb stranice. Određuje stil u kom se *HTML* elementi prikazuju.

JavaScript je skriptni, imperativni programski jezik visokog nivoa. Služi za komunikaciju sa *back-end* stranom sajta, reagovanje na događaje, proveru podataka pre slanja serveru, modifikaciju i čitanje sadržaja *HTML* elementa i tako dalje.

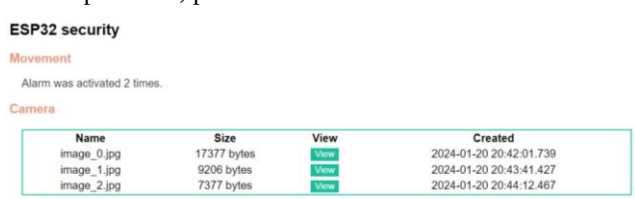
Izgled *front-end* dela veb sajta

Na slici 6. prikazan je deo veb stranice, zadužen za prikaz logova, izbor log nivoa i podešavanja kapaciteta bafera za skladištenje poruka.



Slika 6. Veb stranica log deo

Bira se jedan od šest log nivoa (*none, error, warning, info, debug, verbose*) i jedan od četiri moguća kapaciteta (256, 512, 1024, 2048 bajtova). Podatak o promeni se pritiskom na odgovarajuće dugme šalje na *back-end*. Pritiskom na dugme *Show log file* ispisuju se log poruke iz baze podataka, posredstvom serverske strane.



Slika 7. Veb stranica deo vezan na kameru

Na slici 7. prikazana je tabela sa veb stranice koja sadrži informacije o fotografijama koje su dobijene sa *ESP32-CAM* modula. Pritiskom na dugme *view* ispod tabele prikaže se izabrana slika.

5. ZAKLJUČAK

Cilj rada bio je implementacija mehanizma za optimizovano logovanje i njegovo testiranje na konkretnom embeded sistemu. Ideja je bila unaprediti log mehanizam, tako da log poruke zauzmu što manje mesta u memoriji. Tokom testiranja funkcija i celokupnog sistema, kao ni tokom korišćenja veb servera nije se naišlo na probleme ili greške.

U nekim budućim nadogradnjama, bilo bi interesantno da se umesto ručnog dodavanja jedistvenog broja poruke koriste *hash* funkcije, što bi podrazumevalo prelazak sa programskog jezika C na C++. Ukoliko bi *hash* funkcija bila *constexp* tipa, računanje *string hash-a* bilo bi moguće uraditi u toku kompajliranja. U tom slučaju *string hash* vrednost bi predstavljala identifikacioni broj poruke [5].

Takođe, moguće je nadograditi funkcije za logovanje tako da se osim na veb server podaci loguju i na *SD* karticu. Ovakvo rešenje bilo bi dobro kod uređaja koji nemaju pristup internetu.

Server se može unaprediti dodavanjem više funkcija za upravljanje logovima, slikama i drugim podacima.

Nadogradnja embeded sistema moguća je u pogledu dodavanja većeg broja senzora za kontrolu atmosferskih uslova ili senzora za detekciju nekih spoljašnjih faktora poput požara.

6. LITERATURA

[1] https://www.ascii-code.com/, pregledano januar 2024.

[2] https://github.com/espressif/esp-idf, pregledano januar 2024.

[3] Shah, Dhruti . Node.JS Guidebook. BPB Publications, 2018.

[4] https://www.elektronika.ftn.uns.ac.rs/umrezeni-embeded-sistemi/wp-content/uploads/sites/176/2018/03/UES-04-Front-end-i-Back-end.pdf , pregledano januar 2024.

[5] https://www.youtube.com/watch?v=Dt0vx-7e_B0, pregledano januar 2024

Kratka biografija:

Jelena Marinković rođena je 1997. godine. Diplomski rad na Fakultetu tehničkih nauka iz oblasti Elektrotehnika i računarstvo – Embeded sistemi i algoritmi odbranila je 2020. god.

kontakt: jelenamarpet@gmail.com