



УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У
НОВОМ САДУ



Немања Раковић

**Једно решење прилагођења ФСМ
библиотеке Линукс заснованој
платформи**

ДИПЛОМСКИ РАД
- Основне академске студије -

Нови Сад, 2020



КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, РБР:	
Идентификациони број, ИБР:	
Тип документације, ТД:	Монографска документација
Тип записа, ТЗ:	Текстуални штампани материјал
Врста рада, ВР:	Завршни (Bachelor) рад
Аутор, АУ:	Немања Раковић
Ментор, МН:	проф. др Илија Башичевић
Наслов рада, НР:	Једно решење прилагођења ФСМ библиотеке Линукс заснованој на платформи
Језик публикације, ЈП:	Српски / латиница
Језик извода, ЈИ:	Српски
Земља публиковања, ЗП:	Република Србија
Уже географско подручје, УГП:	Војводина
Година, ГО:	2020
Издавач, ИЗ:	Ауторски репринт
Место и адреса, МА:	Нови Сад; трг Доситеја Обрадовића 6
Физички опис рада, ФО: (поглавља/страна/ цитата/табела/ слика/графика/прилога)	7/23/7/2/13/0/0
Научна област, НО:	Електротехника и рачунарство
Научна дисциплина, НД:	Рачунарска техника
Предметна одредница/Каучне речи, ПО:	ФСМ, комуникациони протоколи, Линукс
УДК	
Чува се, ЧУ:	У библиотеци Факултета техничких наука, Нови Сад
Важна напомена, ВН:	
Извод, ИЗ:	У овом раду представљени су концепт и имплементација прилагођавања ФСМ библиотеке, која се користи за симулацију комуникационих протокола, за рад на Линукс платформи. Главни проблем прилагођавања библиотеке су представљале различите имплементације одређених концепата, попут програмских нити и семафора, на Виндовс и Линукс платформама. Решење је тестирано тако што је један пример који покрива главне функционалности ове библиотеке, покренут на обе платформе и резултати директно упоређени.
Датум прихватања теме, ДП:	
Датум одбране, ДО:	
Чланови комисије, КО:	Председник: <име председника комисије>
Члан:	<име члана комисије>
Члан, ментор:	проф. др Илија Башичевић
	Потпис ментора



KEY WORDS DOCUMENTATION

Accession number, ANO:		
Identification number, INO:		
Document type, DT:	Monographic publication	
Type of record, TR:	Textual printed material	
Contents code, CC:	Bachelor Thesis	
Author, AU:	Nemanja Raković	
Mentor, MN:	PhD Ilija Bašičević	
Title, TI:	One solution for porting the FSM libary to a Linux based system	
Language of text, LT:	Serbian	
Language of abstract, LA:	Serbian	
Country of publication, CP:	Republic of Serbia	
Locality of publication, LP:	Vojvodina	
Publication year, PY:	2020	
Publisher, PB:	Author's reprint	
Publication place, PP:	Novi Sad, Dositeja Obradovica sq. 6	
Physical description, PD: (chapters/pages/ref.tables/pictures/graphs/appendixes)	7/23/7/2/13/0/0	
Scientific field, SF:	Electrical Engineering	
Scientific discipline, SD:	Computer Engineering, Engineering of Computer Based Systems	
Subject/Key words, S/KW:	FSM, communication protocol, Linux	
UC		
Holding data, HD:	The Library of Faculty of Technical Sciences, Novi Sad, Serbia	
Note, N:		
Abstract, AB:	This thesis presents the concept and the implementation of porting the FSM library, which simulates communication protocols, to a Linux based platform. The main problems in the process of porting the library were caused by the different implementations of certain programming concepts such as threads and semaphores, on the Linux and Windows platforms. The results of this thesis were tested by using an example that covers most important functionalities of the library, and running it on both platforms and directly comparing the results.	
Accepted by the Scientific Board on, ASB:		
Defended on, DE:		
Defended Board, DB: President:	<ime predsednika komisije>	
Member:	<ime člana komisije>	Menthor's sign
Member, Mentor:	PhD Ilija Bašičević	

Zahvalnost

Ovom prilikom se zahvaljujem mentoru prof. dr Iliji Bašičeviću kao i MSc Milošu Pilipoviću za podršku i smernice tokom izrade ovog rada. Takođe se zahvaljujem porodici i prijateljima koji su uz mene i podržavaju moj napredak.

SADRŽAJ

1.	Uvod.....	1
2.	FSM biblioteka	2
2.1	FiniteStateMachine klasa	3
2.2	FSMSystem klasa.....	3
2.3	LogFile klasa.....	3
2.4	Interna organizacija FSM biblioteke.....	4
2.4.1	Kernel	5
2.4.2	Odnos FiniteStateMachine i FSMSystem klasa	6
2.5	Model projekta koji koristi FSM biblioteku.....	7
3.	Koncept rešenja.....	8
3.1	Funkcionalnosti koje postoje na obe platforme	8
3.1.1	Tipovi podataka	8
3.1.2	Niti	9
3.1.3	Semafori.....	9
3.2	Funkcionalnosti koje ne postoje na obe platforme	10
4.	Programsko rešenje.....	11
4.1	Radno okruženje.....	11
4.2	Preprocesorske direktive	11
4.3	Tipovi podataka.....	12
4.4	Primena rešenja kroz primer	12
4.4.1	Opis klase.....	12
4.4.2	Stvaranje semafora i njegova deaktivacija.....	13
4.4.3	Uvećavanje vrednosti semafora	14

4.4.4	Sinhronizacija i semafori	14
5.	Evaluacija.....	16
5.1	AutoExample klasa	16
5.1.1	Inicijalizacija.....	16
5.1.2	Start metoda	17
5.1.3	Promene stanja.....	17
5.2	Poređenje rezultata između dve platforme	18
5.2.1	Pokretanje programa	18
5.2.2	Vremenska kontrola.....	19
5.2.3	Zaustavljanje programa	20
5.2.4	Log datoteka	21
6.	Zaključak	22
7.	Literatura.....	23

SPISAK SLIKA

Slika 2.1 Primer automata sa konačnim brojem stanja	2
Slika 2.2 Interna organizacija biblioteke	4
Slika 2.3 Struktura kernela	5
Slika 2.4 Odnos FiniteStateMachine i FSMSystem klasa	6
Slika 2.5 Model jednog jednostavnog projekta	7
Slika 5.1 Pokretanje programa na Linuks platformi	18
Slika 5.2 Pokretanje programa na Windows platformi	18
Slika 5.3 Vremenska kontrola na Linuks platformi	19
Slika 5.4 Vremenska kontrola na Windows platformi	19
Slika 5.5 Zaustavljanje programa na Linuks platformi	20
Slika 5.6 Zaustavljanje programa na Windows platformi	20
Slika 5.7 Log datoteka na Linuks platformi	21
Slika 5.8 Log datoteka na Windows platformi	21

SPISAK TABELA

Tabela 3.1 Ekvivalentne funkcije za korišćenje niti	8
Tabela 3.2 Ekvivalentne funkcije za korišćenje semafora	8

SKRAĆENICE

FSM	- <i>Finite State Machine</i> , Automat sa konačnim brojem stanja
GNU GCC	- skup prevodioca za razne programske jezike
POSIX	- <i>Portable Operating System Interface</i> , porodica povezanih standarda za održavanje kompatibilnosti između operativnih sistema
TCP	- <i>Transmission Control Protocol</i> , transmisioni kontrolni protokol
TPKT	- <i>ISO Transport Service on top of the TCP</i> , enkapsulacija TCP protokola
UDP	- <i>User Datagram Protocol</i> , protokol koji obezbeđuje osnovne funkcije transportnog sloja OSI modela

1. Uvod

FSM biblioteka je namenjena za simulaciju i implementaciju telekomunikacionih protokola, tj. pravila koja definišu razmenu poruka preko neke mreže. Njena trenutna glavna primena je u oblasti obrazovanja, gde studentima pruža priliku ne samo da se na što bližem nivou susretu sa mrežnim protokolima, već i da sami učestvuju u procesu osmišljavanja i njihovoj realizaciji.

Biblioteka funkcioniše i aktivno se koristi na Windows operativnom sistemu. Iako je biblioteka pisana u C++ programskom jeziku, koji je podržan na obe platforme, ona se u svom prvobitnom obliku ne može koristi na Linuks platformi. Razlozi zašto je to slučaj, kao i proces identifikacije i pristup rešavanju problema, su glave stavke na kojima je ovaj rad zasnovan.

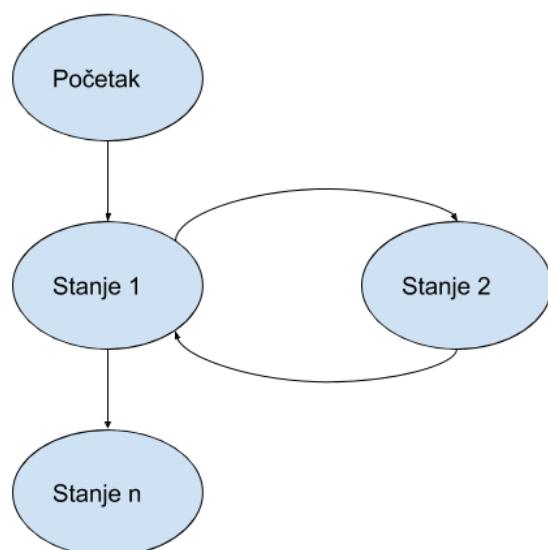
Rad je podeljen na četiri glavne celine, od kojih prva opisuje već postojeću FSM biblioteku, koncepte na kojima je zasnovana kao i njenu funkcionalnost. Rad će se potom osvrnuti na fundamentalne razlike između Windows i Linuks platforme, i različite implementacije određenih koncepata, koji otežavaju korišćenje ne samo ove biblioteke nego i dosta drugih programa na obe platforme. Ukratko će biti opisani drugačiji pristupi rešavanju nekih od najčešćih problema koji se javljaju.

Poglavlje o samom programskom rešenju, se direktno bavi implementacijom i detaljnijem objašnjenju funkcija jedne klase. Baš ta klasa je izabrana jer služi kao dobar primer gde se različita rešenja zajedno koriste da obezbede potrebnu funkcionalnost.

Poslednju celinu čini evaluacija rešenja. Pošto je cilj da se dostigne ekvivalentna funkcionalnost biblioteke kao na Windows platformi, biće upoređena upotreba biblioteke na obe platforme kroz nekoliko demonstrativnih primera.

2. FSM biblioteka

FSM biblioteka se tokom godina proširivala i unapređivala, i danas se primarno koristi u edukativne svrhe. Njen cilj je da obezbedi radno okruženje za implementaciju komunikacionih protokola tako što povezuje konačne automate preko sistema automata (FSM sistem) koji rukuje resursima potrebnim za rad protokola. FSM ili automat sa konačnim brojem stanja je apstraktna mašina koja se u određenom trenutku može nalaziti samo u jednom od prethodno definisanih stanja. Definisana je listom svih mogućih stanja, početnim stanjem, kao i događajima koji dovode do promene stanja. Automati sa konačnim brojem stanja imaju široke primene i nalaze se u skoro svakom nešto složenijem digitalnom sistemu.



Slika 2.1 Primer automata sa konačnim brojem stanja

Dve ključne i najvažnije klase na kojima se zasniva FSM biblioteka su `FSMSys`tem i `FiniteStateMachine` koje su u međusobnom stanju simbioze, i jedna bez druge nemaju puno smisla.

2.1 FiniteStateMachine klasa

Svaki automat je realizovan tako što nasleđuje i proširuje baznu klasu `FiniteStateMachine`. U toj klasi su definisane virtuelne metode koje je potrebno implementirati, one su:

```
MessageInterface *GetMessageInterface( uint32 id );
void SetDefaultHeader( uint8 infoCoding );
uint8 GetMbId();
uint8 GetAutomate();
void SetDefaultFSMData();
void NoFreeInstances();
void Initialize();
```

Detaljnije objašnjenje ovih funkcija je dostupno u pratećoj dokumentaciji biblioteke.

2.2 FSMSys klasa

Ova klasa predstavlja sistem mašina sa konačnim brojem stanja. Njena svrha je da međusobno poveže individualne automate time što upravlja svim resursima koji su im potrebni za funkcionisanje sistema.

Tipični koraci u inicijalizaciji:

- Kreiranje samog sistema automata preko konstruktora
- Kreiranje i inicijalizacija individualnih automata
- Ubacivanje automata u FSM sistem
- Započinjanje sistema za praćenje događaja

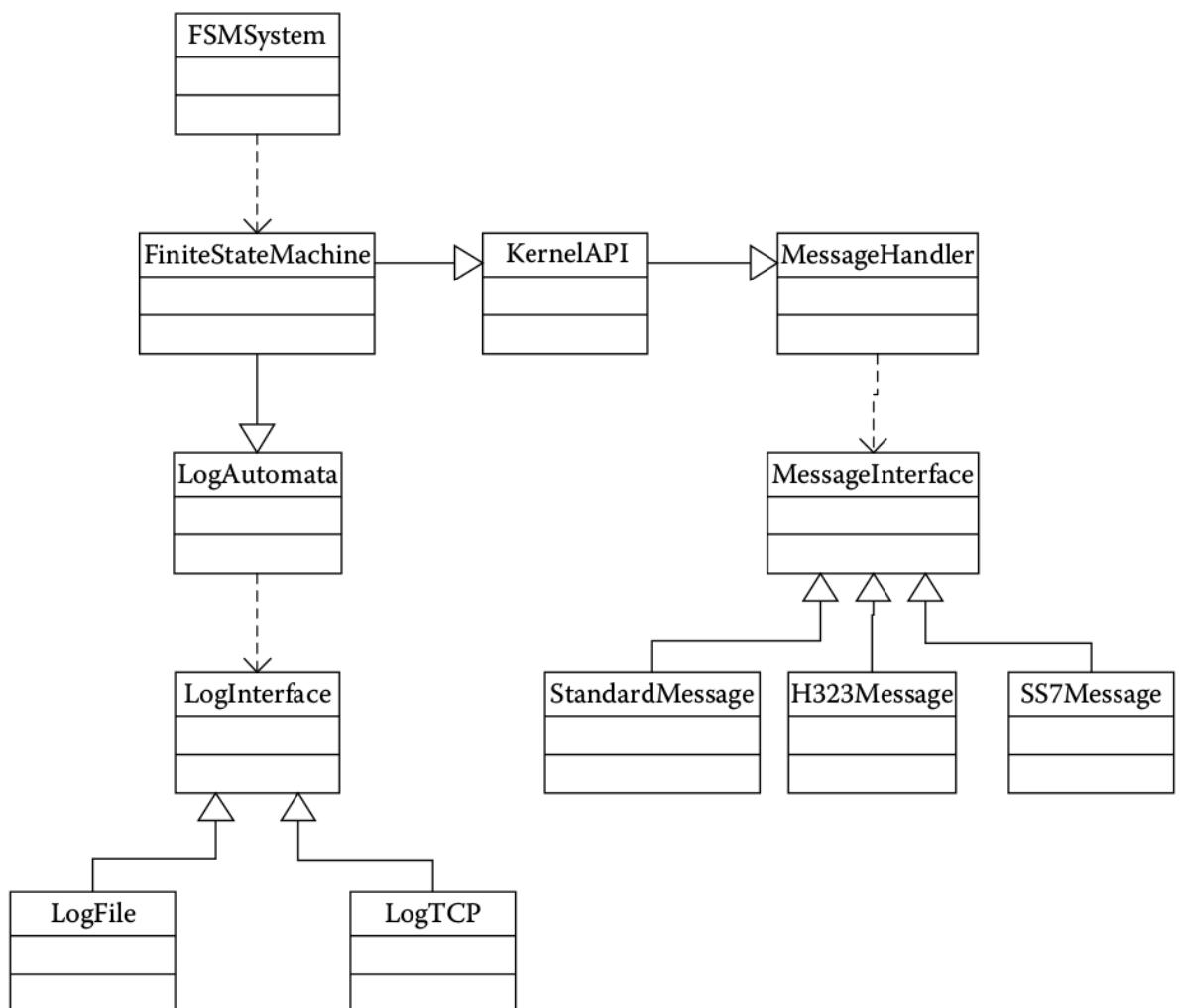
2.3 LogFile klasa

Sistem za praćenje događaja obezbeđuje mogućnost vođenja evidencije o poslatim porukama i sistemskih događaja kao što su promene stanja automata i vremenska kontrola. Događaji se automatski zapisuju u datoteku stvorenu pri kreiranju FSM sistema. Postoji i mogućnost korišćenja „log.ini” datoteke preko koje se mogu definisati parametri koji olakšavaju čitanje i tumačenje zabeleženih događaja.

2.4 Interna organizacija FSM biblioteke

MessageInterface je apstraktna klasa za rukovanje porukama. Nju nasleđuje MessageHandler koja ima dodatnu mogućnost za dodavanje i uklanjanje parametara, kao i formiranje poruka.

Klasa KernelAPI je izvedena iz MessageHandler klase, pruža spregu za kernel funkcije i omogućava rukovanje memorijom, vremensku kontrolu i razmenu poruka. Način na koji su povezane sa klasama opisanim na prethodnoj stranici se može videti na dijagramu ispod.



Slika 2.2 Interna organizacija biblioteke

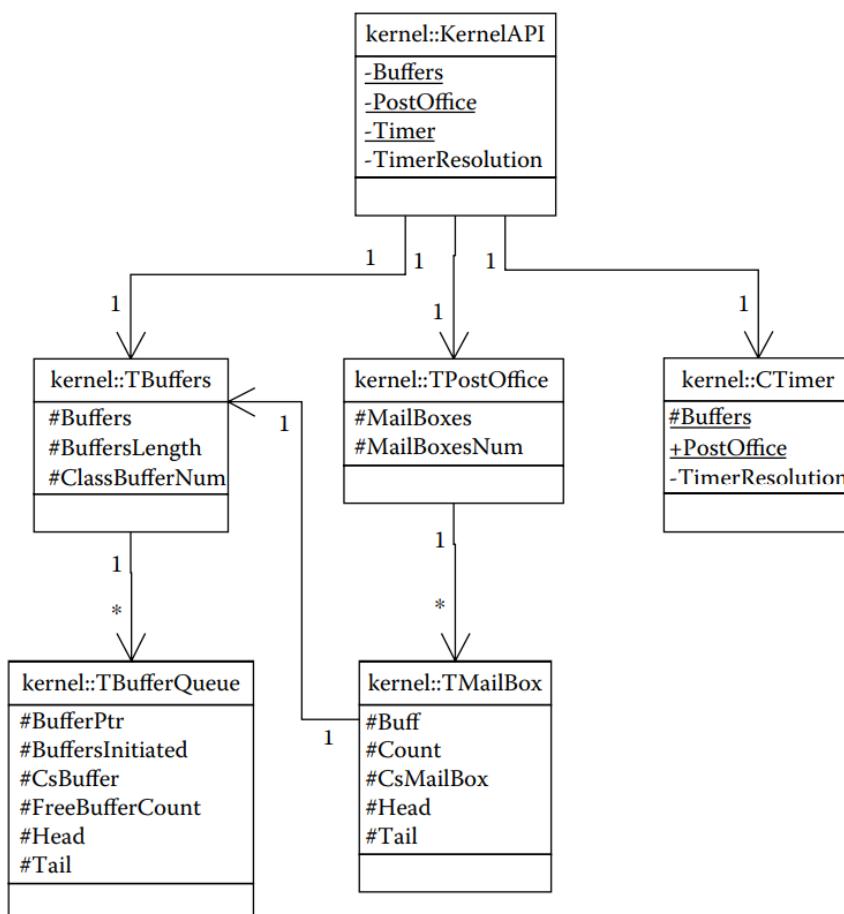
2.4.1 Kernel

Svrha KernelAPI klase je da klasi FiniteStateMachine pruži što lakši pristup potrebnim resursima nezavisno od implementacije samog kernela. Tri glavne komponente kernela su rukovaoci za memoriju, poruke i vreme.

Rukovanje memorijom je obezbeđeno kroz klasu TBuffers koja sadrži niz redova sa baferima različitih veličina. Klasa TBuffersQueue upravlja baferima određene veličine.

Vremenska kontrola je omogućena pomoću CTimer klase. Njen glavni zadatak je da upravlja tajmerima, što uključuje njihovo započinjanje, zaustavljanje i skladištenje. Lista aktivnih tajmera je implementirana kao delta lista, tj. struktura podataka izvedena iz reda sa prioritetom.

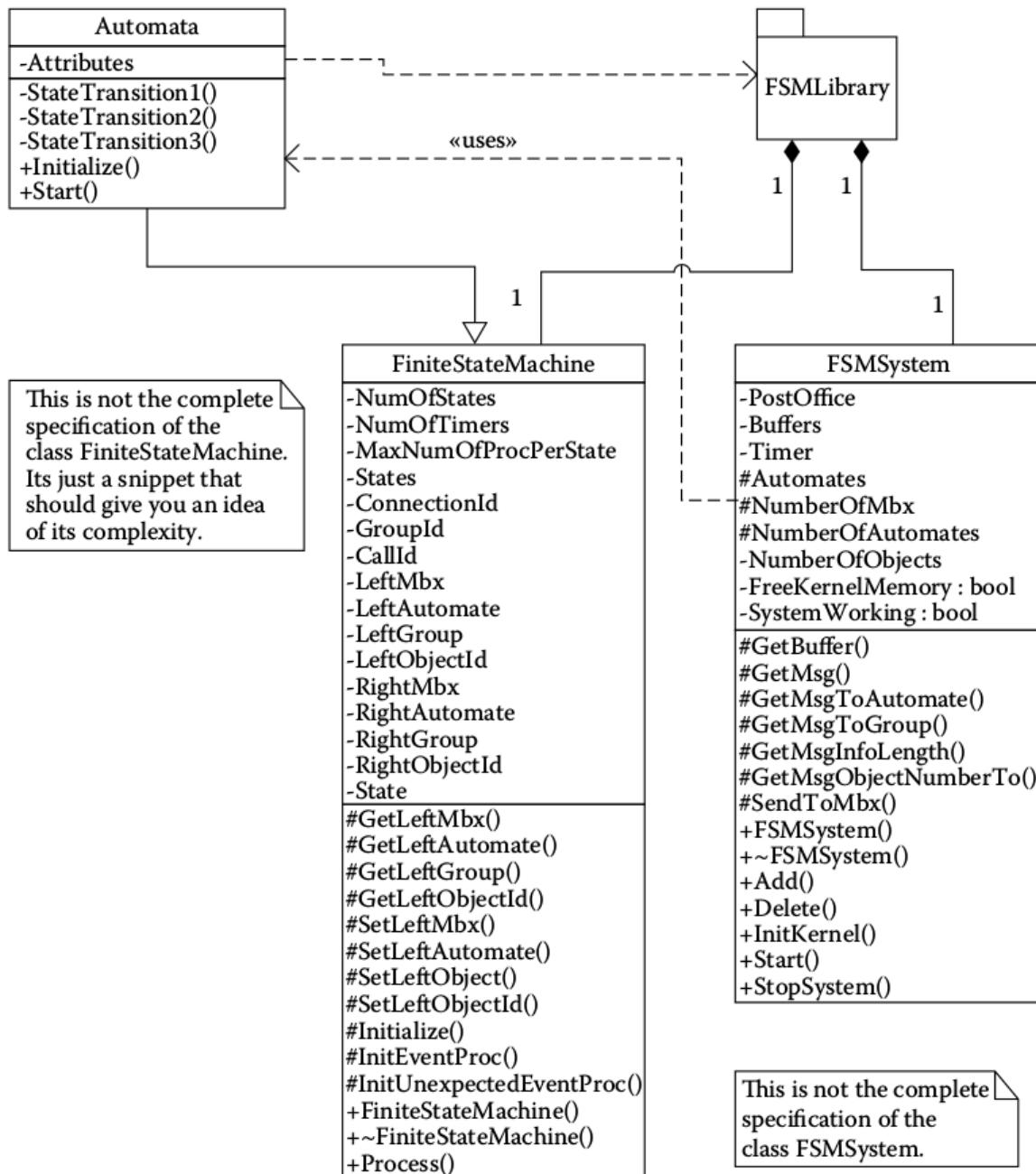
Kontrola porukama proizlazi iz klase TPostOffice koja skladišti niz pokazivača na odgovarajuće poštanske sandučiće. Poštansko sanduče je implementirano kao instanca klase TMailBox koja je veoma slična klasi TBuffersQueue sa bitnom razlikom što TMailBox obezbeđuje pristup deljenim resursima od strane dva procesa. To se postiže funkcijama MbxLock() i MbxUnlock() koje zaključavaju, odnosno otključavaju pristup poštanskom sandučiću.



Slika 2.3 Struktura kernela

2.4.2 Odnos FiniteStateMachine i FSMSystem klasa

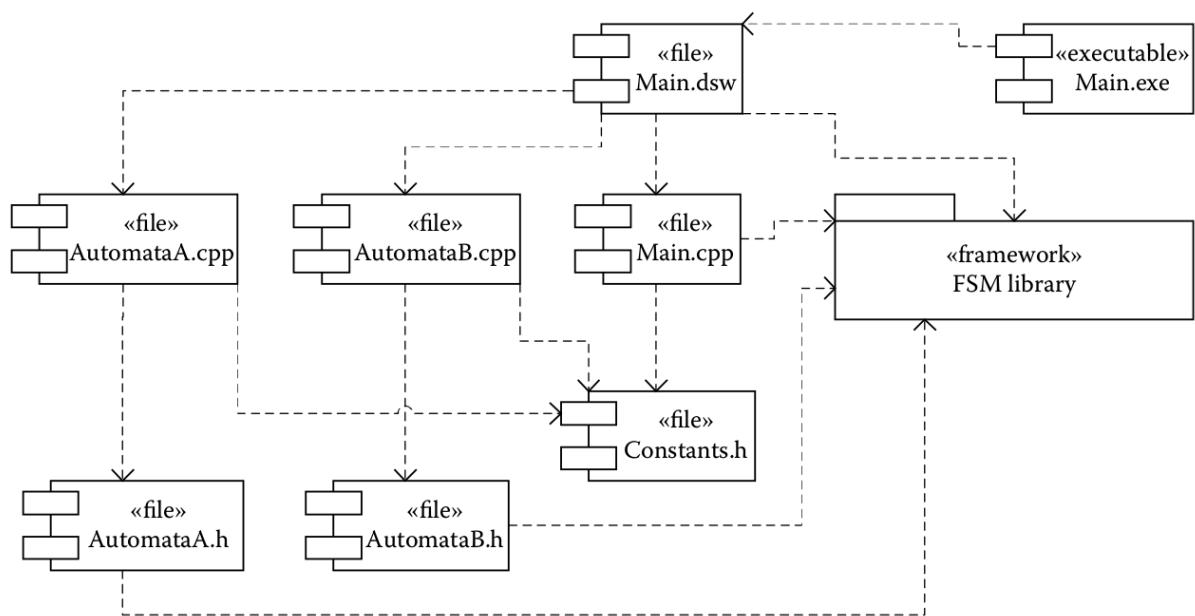
Na slici ispod se može videti pregled nekih od funkcija klase FiniteStateMachine i FSMSystem, i okvirno prikazan način na koji su povezane sa ostatkom biblioteke.



Slika 2.4 Odnos FiniteStateMachine i FSMSystem klasa

2.5 Model projekta koji koristi FSM biblioteku

Na slici je dat primer organizacije jednog jednostavnog projekta. Klase AutomataA i AutomataB svaka nasleđuje klasu FiniteStateMachine i implementira svoje funkcije i time predstavlja novi tip automata. Obe su povezane sa datotekom Constants.h koja sadrži i postavlja vrednosti nekih od parametara za inicijalizaciju. Ovo je tipičan način primene biblioteke i sličan model će se koristiti u poglavlju o evaluaciji rešenja ovog rada.



Slika 2.5 Model jednog jednostavnog projekta

3. Koncept rešenja

FSM biblioteku u prvobitnom stanju nije bilo moguće prevesti za Linuks platformu. Da bi se to omogućilo, i ujedno postiglo njeno očekivano ponašanje, bilo je potrebno doneti dosta izmena, od manjih kao što su prepravke velikih slova u mala, sve do pisanja skroz novih funkcija koje zamenjuju one sa Windows platforme za koje ne postoji Linuks alternativa.

Postoje brojne razlike između dve platforme, ali će fokus biti na problemima koji su relevantni za adaptaciju ove biblioteke. Oni se svode na upotrebu funkcija i funkcionalnosti koje ili isključivo postoje na Windows platformi, ili su implementirane na specifičan način što sprečava njihovo korišćenje na Linuks platformi. Ti problemi će biti podeljeni u dve kategorije, oni za koje postoji ekvivalentna opcija na obe platforme, i oni za koje to nije slučaj.

3.1 Funkcionalnosti koje postoje na obe platforme

3.1.1 Tipovi podataka

Određeni tipovi podataka ili nisu podržani na obe platforme, ili su implementirani na drugačiji način. Većina ovakvih problema je relativno lako rešiva, jer se dosta tipova pod različitim imenima svode na jedan te isti osnovni tip.

3.1.2 Niti

Niti omogućavaju konkurentno izvršavanje koda, tačnije pokretanje različitih delova istovremeno. Windows platforma ima svoju implementaciju niti, dok se na Linuks platformi koristi pthread biblioteka.

Win32	Linux
_beginthread	pthread_attr_init pthread_attr_setstacksize pthread_create
_endthread	pthread_exit
TerminateThread	pthread_cancel
GetCurrentThreadId	pthread_self

Tabela 3.1 Ekvivalentne funkcije za korišćenje niti

3.1.3 Semafori

Semafor je promenljiva koja se koristi da kontroliše pristup deljenim resursima, i igra ključnu ulogu u sinhronizaciji. Windows platforma ima svoju implementaciju, dok se na Linuks platformi mogu koristiti ili mutex-i preko pthread biblioteke ili preko POSIX semafora. U tabeli ispod se mogu videti analogne operacije između tri pristupa.

Win32	pthread Linux	POSIX
CreateSemaphore	pthread_mutex_init(&(token)->mutex, NULL) pthread_cond_init(&(token)->condition, NULL)	sem_init
CloseHandle (semHandle)	pthread_mutex_destroy(&(token->mutex)) pthread_cond_destroy(&(token->condition))	sem_destroy
ReleaseSemaphore(semHandle, 1, NULL)	pthread_cond_signal(&(token->condition))	sem_post
WaitForSingleObject(semHandle, INFINITE) WaitForSingleObject(semHandle, timelimit)	pthread_cond_wait(&(token->condition), &(token->mutex)) pthread_cond_timedwait(&(token->condition), &(token->mutex))	sem_wait sem_trywait

Tabela 3.2 Ekvivalentne funkcije za korišćenje semafora

3.2 Funkcionalnosti koje ne postoje na obe platforme

Ovakve funkcije i funkcionalnosti predstavljaju mnogo veći problem, i zahtevaju drugačiji pristup, što u pojedinim slučajevima može da dovede do situacije gde je potrebno ne samo obezbediti određenu funkcionalnost nego i promeniti i ostatak koda da bi implementacija uopšte bila moguća. To nije uvek slučaj, i često se željena funkcionalnost može postići kombinacijom već postojećih funkcija.

Kao primer se može uzeti funkcija `WaitForMultipleObjects`, koja se koristiti u FSM biblioteci, ali se ne nalazi u tabeli 1, iz razloga što ne postoji njen implementacija na Linuks platformi. Rešenje ovog specifičnog problema u kontekstu FSM biblioteke je dato u sledećem poglavlju.

4. Programsко rešenje

4.1 Radno okruženje

Za svrhu implementacije programskog rešenja, izabrana je Ubuntu 16.04 distribucija Linuks operativnog sistema zbog što stabilnijeg radnog okruženja. Prevodilac GNU GCC Compiler je korišćen u okviru CodeBlocks razvojnog okruženja koje je izabrano radi lakšeg pranja i otkrivanja grešaka što se pokazalo kao veoma korisno u procesu prilagođavanja biblioteke zbog brojnih sitnijih grešaka.

4.2 Preprocesorske direktive

Preprocesorske directive su jedan od ključnih koncepta koji su omogućili lakšu i jasniju upotrebu i implementaciju specifičnih funkcionalnosti koje su usko vezane za platformu na kojoj se program izvršava. Time je otklonjena potreba da postoje dve odvojene verzije biblioteke za dve različite platforme.

Koristi se uslovno izvršavanje koda kroz `#ifdef` i `#endif` direktive zajedno sa makroima `_linux_` koji predstavlja Linuks platformu i `WIN32` koji predstavlja Windows platformu, najčešće u sledećem obliku.

```
#ifdef __linux__  
    // kod za linux platformu  
#elif WIN32  
    // kod za windows platformu
```

4.3 Tipovi podataka

Problem tipova podataka koji nisu podržani na obe platforme je rešen time što su potrebni Windows tipovi redefinisani na Linuks platformi, tako se omogućava korišćenje nekih od istih funkcija sa Windows platforme.

```
typedef const char* LPCWSTR;
typedef char* LPWSTR;
typedef uint8 BYTE;
typedef uint16 WORD;
typedef uint32 DWORD;
```

LPCWSTR na Windows platformi predstavlja tridesetdvobitni pokazivač na konstantan string naspram LPWSTR koji nije konstantan.

4.4 Primena rešenja kroz primer

U cilju jasnijeg opisa upotrebe različitih pristupa prilagođavanja koda za Linuks platformu, detaljno će biti opisane funkcije klase TPostOffice iz postOffice.cpp datoteke. Ova klasa je izabrana jer predstavlja jedinstven primer gde je neophodno da skoro svaka funkcija bude prilagođena za Linuks platformu i na jasan način pokazuje različite metode koje su korišćene.

4.4.1 Opis klase

TPostOffice klasa, se može zamisliti kao pošta koja je zadužena za razmenu poruka. Ona poseduje poštanske sandučiće (TmailBox), i u njih raspoređuje odgovarajuće poruke koje stižu u poštu.

Da bi se omogućilo jasno praćenje koda, date su neke definicije iz postOffice.h datoteke. Definicije nizova semafora od 255 članova su semArray i semHandle respektivno na Linuks i Windows platformi, što predstavlja maksimalan broj poštanskih sandučića.

```
#define MAX_NUM_OF_MAILBOXES 255
...
#ifndef __linux__
sem_t semArray[MAX_NUM_OF_MAILBOXES];
#else
HANDLE semHandle[MAX_NUM_OF_MAILBOXES];
#endif
```

4.4.2 Stvaranje semafora i njegova deaktivacija

Slede dva primera gde postoje jednostavne alternative za Windows funkcije koji postižu sličan efekat, iako je sama implementacija semafora različita između dve platforme, za potrebe ove biblioteke postignuta funkcionalnost je zadovoljavajuća.

Konstruktor klase TPostOffice:

```
TPostOffice::TPostOffice(uint8 mailBoxesNum, TBuffers *buffers)
{
    ...
    //create mail boxes needed for system
    for (i=0; i<mailBoxesNum; i++)
    {
        #ifdef __linux__
            sem_init(&semArray[i], 0, 0);
        #else
            semHandle[i] = CreateSemaphore( NULL, 0, 10000, NULL);
            if( semHandle[i] == NULL) ThrowError( TErrorObject(...));
        #endif
    }
    ...
}
```

Analogno konstrukturu, situacija je slična i kod destruktora, i postoji

```
#ifdef __linux__
    sem_destroy(&semArray[i]);
#else
    CloseHandle(semHandle[i]);
#endif
```

4.4.3 Uvećavanje vrednosti semafora

Funkcija Add dodaje poruku u odgovarajuće sanduče i povećava vrednost semafora za jedan. Funkcije sem_post i ReleaseSemaphore se skoro identično ponašaju sa razlikom da ReleaseSemaphore može da uveća vrednost semafora za proizvoljan broj.

```
void TPostOffice::Add( uint8 mailBoxID, uint8 *info)
{
    ...
    MailBoxes[mailBoxID]->Add( info);
    #ifdef __linux__
    sem_post(&semArray[mailBoxID]);
    #else
    ReleaseSemaphore(semHandle[mailBoxID], 1, NULL);
    #endif
}
```

4.4.4 Sinhronizacija i semafori

Cilj sinhronizacije u klasi TPostOffice je da se spreče bespotrebni pokušaji dobijanja poruke dok ni jedna poruka nije stigla ni u jedno poštansko sanduče. Biće sagledana funkcija GetSync, gde istu funkcionalnost nije moguće postići jednostavnom upotrebom ekvivalentne funkcije, iz razloga što ona ne postoji.

Windows funkcija WaitForMultipleObjects (MailBoxesNum, semHandle, FALSE, INFINITE) čeka dok se jedan ili svi od njih navedenih objekata ne nađu u signaliziranom stanju, što u slučaju njene primene u ovoj biblioteci znači čekanje dok se ne pojavi nova poruka u jednom od sandučića. Nakon njenog uspešnog izvršavanja, vraća informaciju o sandučiću koje je primilo poruku. Pošto ekvivalentna funkcija ne postoji u Linuksu, ista funkcionalnost mora biti postignuta na drugi način. Funkcija je pozvana sa parametrima:

MailBoxesNum – broj sandučića

semHandle – niz sandučića koji su posmatrani

FALSE – da li svi događaji moraju da budu izvršeni, u ovom slučaju ne

INFINITE – dužina čekanja, u ovom slučaju beskonačna

Zbog navedenih pozivnih parametara Windows funkcije, u ovom slučaju se koristi neblokirajuća funkcija sem_trywait() koja pokušava da umanji vrednost semafora, u kombinaciji sa while petljom, da bi željena funkcionalnost bila postignuta na Linuks platformi.

```
uint8 *TPostOffice::GetSync( uint8 &mailBoxID)
{
    #ifdef __linux__
    int res = -1;
    int i = 0;
    while(res != 0)
    {
        i=0;
        for(i=0;i<MailBoxesNum;i++)
        {
            res = sem_trywait(&semArray[i]);
            if(res==0)
        }
        mailBoxID = i;
        break;
    }
    #else
        DWORD dwWaitResult = WaitForMultipleObjects (MailBoxesNum,semHandle,
        FALSE,INFINITE);
        mailBoxID = (uint8)(dwWaitResult-WAIT_OBJECT_0);
    #endif

    #ifndef __NO_PARAMETER_CHECK__
    if(mailBoxID >= MailBoxesNum) ThrowError( TErrorObject( __LINE__, __FILE__,
    0x01010200));
    #endif

    return (MailBoxes[mailBoxID]->Get());
}
```

5. Evaluacija

Evaluacija programskog rešenja je izuzetno bitan korak u prilagođavanju biblioteke i predstavlja konkretan način da se proveri do koje mere je ono uspešno.

5.1 AutoExample klasa

Izabrana je klasa AutoExample, iz razloga što je baš to klasa koja se koristi u okviru računarskih vežbi na fakultetu za upoznavanje studenata sa bibliotekom kao i prikaz njenih sposobnosti.

5.1.1 Inicijalizacija

Metoda Initialize() postavlja početno stanje u AUTO_STATE0, definiše uslove pod kojima će doći do promene stanja, i upravlja brojačem vremena.

```
void AutoExample::Initialize()
{
    SetState(AUTO_STATE0);
    SetDefaultFSMData();
    InitEventProc(AUTO_STATE0, MSG_CHANGE_STATE,
        (PROC_FUN_PTR)&AutoExample::S0_ChangeState);
    InitEventProc(AUTO_STATE1, TIMER1_EXPIRED, (PROC_FUN_PTR)&AutoExample::S1_ChangeSt
        ate);
    InitTimerBlock(TIMER1_ID, TIMER1_COUNT, TIMER1_EXPIRED);
    StartTimer(TIMER1_ID);
}
```

5.1.2 Start metoda

U okviru Start metode, da bi se pokazala i ta mogućnost, automat sam sebi šalje poruku koja mu javlja da promeni stanje.

```
void AutoExample::Start()
{
    printf("Starting automate: %d \n", GetObjectId());
    PrepareNewMessage(0x00, MSG_CHANGE_STATE);
    SetMsgToAutomate(AUTOEXAMPLE_FSM);
    SetMsgObjectNumberTo(GetObjectId()); // Salje sam sebi.
    SendMessage(AUTOEXAMPLE_MBX_ID);

}
```

5.1.3 Promene stanja

Ove funkcije se pozivaju nakon što dođe do promene stanja. S0_ChangeState() menja stanje u AUTO_STATE1, dok S1_ChangeState() ostaje u istom stanju.

```
void AutoExample::S0_ChangeState()
{
    printf("AutoExample[%d]::S0_ChangeState() - receive message !\n",
GetObjectId());
    SetState(AUTO_STATE1);
    printf("Current state is: %d \n", GetState());

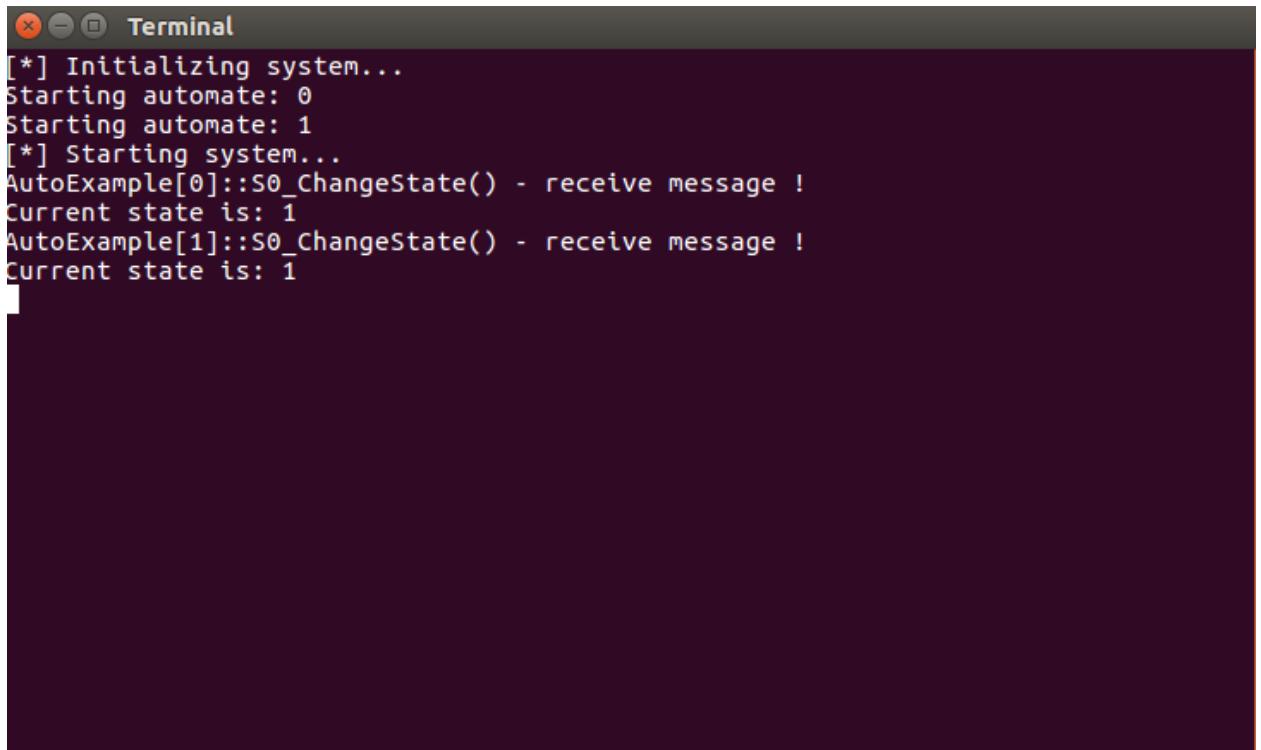
}

void AutoExample::S1_ChangeState()
{
    printf("AutoExample[%d]::S1_ChangeState() - istekla je vremenska kontrola !\n",
GetObjectId());
    SetState(AUTO_STATE1);
    StopTimer(TIMER1_ID);
}
```

5.2 Poređenje rezultata između dve platforme

5.2.1 Pokretanje programa

Poziv Start() metode, a kao posledica se poziva i S0ChangeState().



```
[*] Initializing system...
Starting automate: 0
Starting automate: 1
[*] Starting system...
AutoExample[0]::S0_ChangeState() - receive message !
Current state is: 1
AutoExample[1]::S0_ChangeState() - receive message !
Current state is: 1
```

Slika 5.1 Pokretanje programa na Linuks platformi

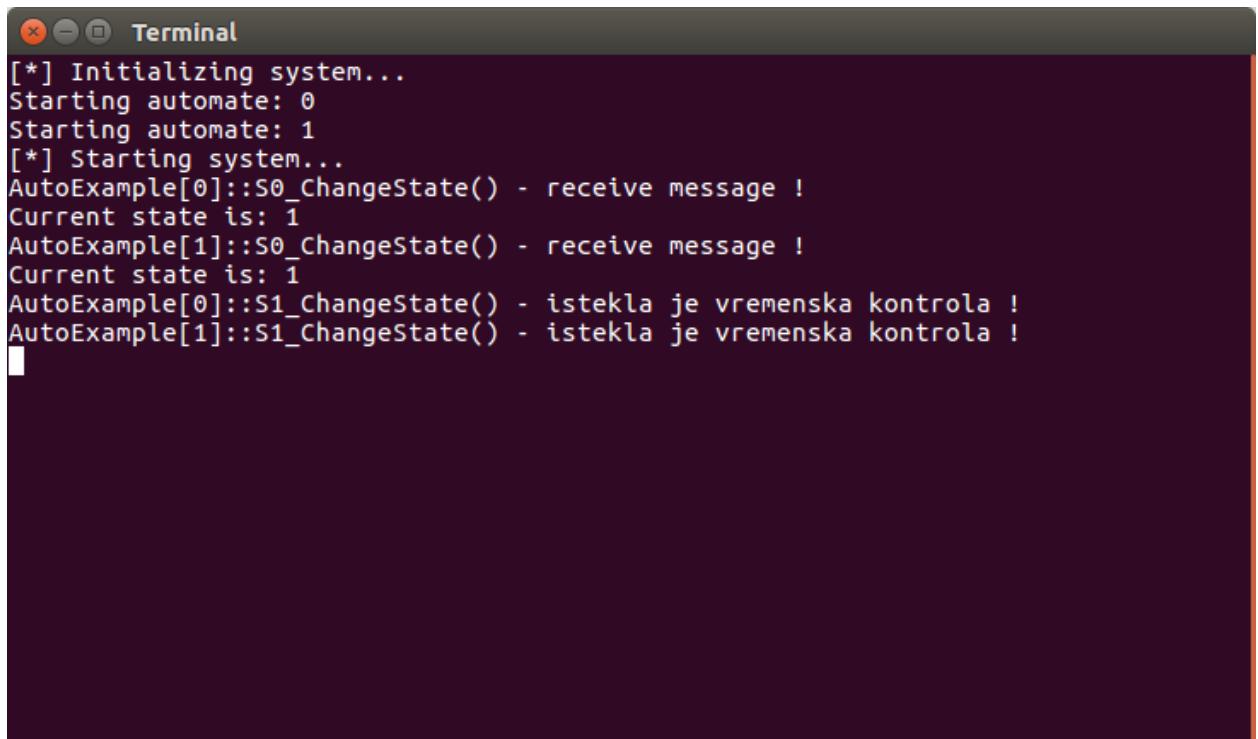


```
[*] Initializing system...
Starting automate: 0
Starting automate: 1
[*] Starting system...
AutoExample[0]::S0_ChangeState() - receive message !
Current state is: 1
AutoExample[1]::S0_ChangeState() - receive message !
Current state is: 1
```

Slika 5.2 Pokretanje programa na Windows platformi

5.2.2 Vremenska kontrola

Po isteku brojača vremena, stanje se postavlja u S1.



```
[*] Initializing system...
Starting automate: 0
Starting automate: 1
[*] Starting system...
AutoExample[0]::S0_ChangeState() - receive message !
Current state is: 1
AutoExample[1]::S0_ChangeState() - receive message !
Current state is: 1
AutoExample[0]::S1_ChangeState() - istekla je vremenska kontrola !
AutoExample[1]::S1_ChangeState() - istekla je vremenska kontrola !
```

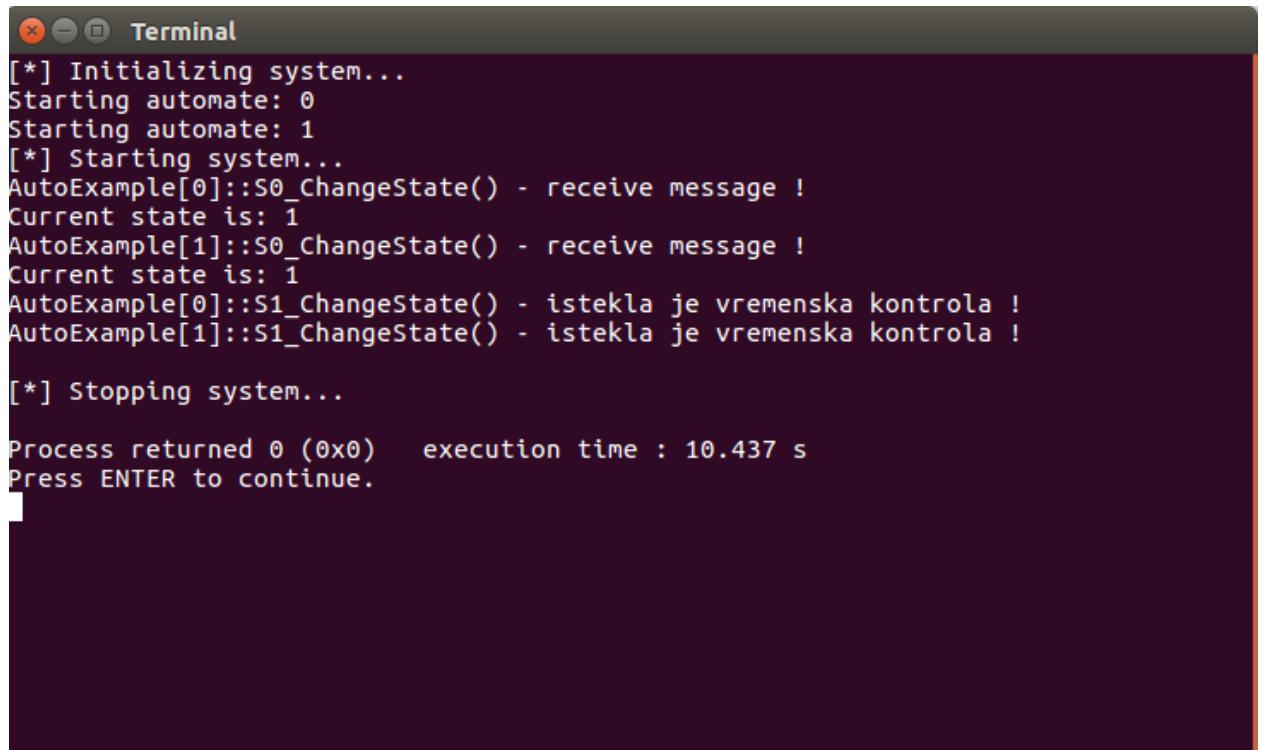
Slika 5.3 Vremenska kontrola na Linuks platformi

```
[*] Initializing system...
Starting automate: 0
Starting automate: 1
[*] Starting system...
AutoExample[0]::S0_ChangeState() - receive message !
Current state is: 1
AutoExample[1]::S0_ChangeState() - receive message !
Current state is: 1
AutoExample[0]::S1_ChangeState() - istekla je vremenska kontrola !
AutoExample[1]::S1_ChangeState() - istekla je vremenska kontrola !
```

Slika 5.4 Vremenska kontrola na Windows platformi

5.2.3 Zaustavljanje programa

Po ručnom unosu sa tastature, alocirani resursi se oslobođaju i dolazi do završetka programa.

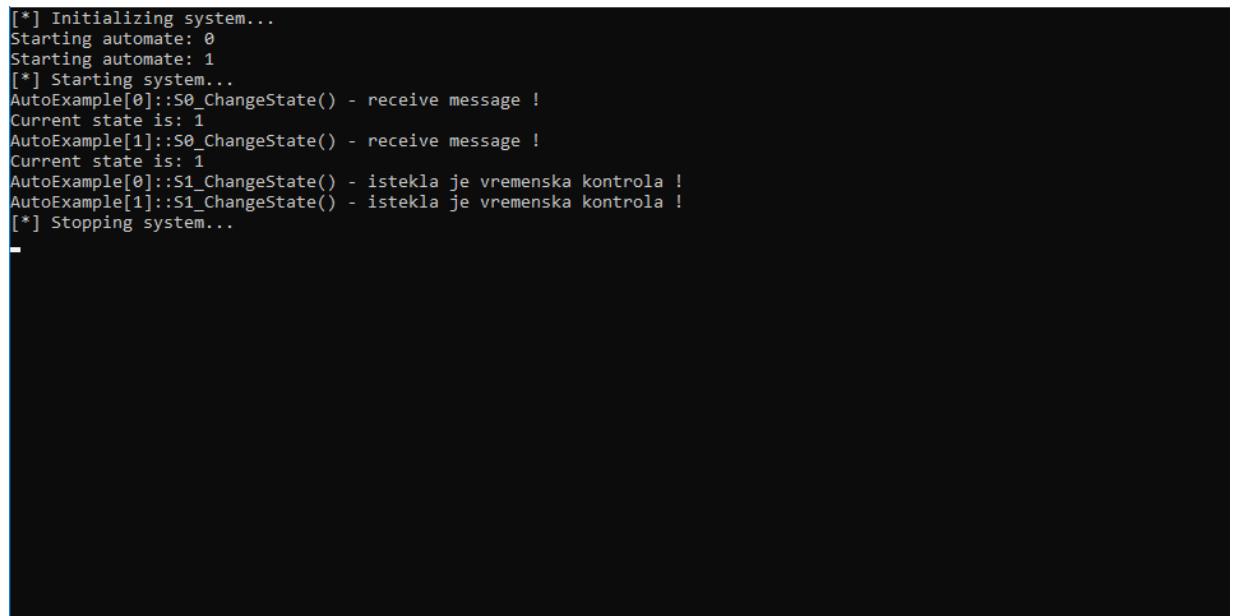


```
[*] Initializing system...
Starting automate: 0
Starting automate: 1
[*] Starting system...
AutoExample[0]::S0_ChangeState() - receive message !
Current state is: 1
AutoExample[1]::S0_ChangeState() - receive message !
Current state is: 1
AutoExample[0]::S1_ChangeState() - istekla je vremenska kontrola !
AutoExample[1]::S1_ChangeState() - istekla je vremenska kontrola !

[*] Stopping system...

Process returned 0 (0x0)   execution time : 10.437 s
Press ENTER to continue.
```

Slika 5.5 Zaustavljanje programa na Linuks platformi



```
[*] Initializing system...
Starting automate: 0
Starting automate: 1
[*] Starting system...
AutoExample[0]::S0_ChangeState() - receive message !
Current state is: 1
AutoExample[1]::S0_ChangeState() - receive message !
Current state is: 1
AutoExample[0]::S1_ChangeState() - istekla je vremenska kontrola !
AutoExample[1]::S1_ChangeState() - istekla je vremenska kontrola !
[*] Stopping system...
```

Slika 5.6 Zaustavljanje programa na Windows platformi

5.2.4 Log datoteka

Prikaz Log datoteke na obe platforme.

```
Start Timer: (0)
Start Timer: (0)
Mon Aug 24 16:53:20 2020
Msg To: (0x00), Automate ID: 0x00000000
MsgFrom: (0x00), Automate ID: 0x00000000
Received Msg: (0x0000), Length: 0 Coding type: 0|
00 00 00 00 | 00 00 00 00 | 00 00 00 00 | 00 00 00 00 | 00
State: 0 -> 1

Mon Aug 24 16:53:20 2020
Msg To: (0x00), Automate ID: 0x00000001
MsgFrom: (0x00), Automate ID: 0x00000001
Received Msg: (0x0000), Length: 0 Coding type: 0
00 00 00 00 | 00 00 01 00 | 00 00 01 00 | 00 00 00 00 | 00 00 00 00 | 00
State: 0 -> 1

Mon Aug 24 16:53:25 2020
Msg To: (0x00), Automate ID: 0x00000000
MsgFrom: (0x00), Automate ID: 0x00000000
Received Msg: (0x0001), Length: 0 Coding type: 0
00 00 00 00 | 01 00 00 00 | 00 00 00 00 | 00 00 ff ff | ff ff 00 00 | 00
Stop Timer: (0)
State: 1 -> 1

Mon Aug 24 16:53:25 2020
Msg To: (0x00), Automate ID: 0x00000001
MsgFrom: (0x00), Automate ID: 0x00000000
Received Msg: (0x0001), Length: 0 Coding type: 0
00 00 00 00 | 01 00 00 00 | 00 00 01 00 | 00 00 ff ff | ff ff 00 00 | 00
Stop Timer: (0)
State: 1 -> 1
```

Slika 5.7 Log datoteka na Linuks platformi

```
$Start Timer: (4)
Start Timer: (4)
Wed Sep 09 07:10:02 2020
Msg To: (0x00), Automate ID: 0x00000000
MsgFrom: (0x00), Automate ID: 0x00000000
Received Msg: (0x0000), Length: 0 Coding type: 0
00 00 00 cd | 00 00 00 00 | 00 00 00 00 | 00 00 cd cd | cd cd 00 00 | 00
State: 0 -> 1

Wed Sep 09 07:10:02 2020
Msg To: (0x00), Automate ID: 0x00000001
MsgFrom: (0x00), Automate ID: 0x00000001
Received Msg: (0x0000), Length: 0 Coding type: 0
00 00 00 cd | 00 00 01 00 | 00 00 01 00 | 00 00 cd cd | cd cd 00 00 | 00
State: 0 -> 1

Wed Sep 09 07:10:07 2020
Msg To: (0x00), Automate ID: 0x00000000
MsgFrom: (0x00), Automate ID: 0x00000000
Received Msg: (0x0001), Length: 0 Coding type: 0
00 00 00 00 | 01 00 00 00 | 00 00 00 00 | 00 00 ff ff | ff ff 00 00 | 00
Stop Timer: (4)
State: 1 -> 1

Wed Sep 09 07:10:07 2020
Msg To: (0x00), Automate ID: 0x00000001
MsgFrom: (0x00), Automate ID: 0x00000000
Received Msg: (0x0001), Length: 0 Coding type: 0
00 00 00 00 | 01 00 00 00 | 00 00 01 00 | 00 00 ff ff | ff ff 00 00 | 00
Stop Timer: (4)
State: 1 -> 1
```

Slika 5.8 Log datoteka na Windows platformi

6. Zaključak

U ovom radu je pokazano jedno rešenje problema prilagođavanja FSM biblioteke za Linuks platformu. Cilj rada je bio da se dostigne isti nivo funkcionalnosti kao na Windows platformi i time pruži veća fleksibilnost i proširi upotreba ove biblioteke, specifično, ali ne samo za studente koji se upoznaju sa mrežnim protokolima.

Rešavanje ovog problema je pre svega zahtevalo detaljnije upoznavanje sa samom bibliotekom i principima na kojima je zasnovana. Ovaj proces je bio znatno olakšan postojanjem veoma detaljne dokumentacije u vidu priručnika pruženom od strane fakulteta [6] i knjige [1] koja pruža ne samo informacije o komunikacionim protokolima nego i detaljima implementacije biblioteke. Nakon dobrog poznавanja same biblioteke bilo je potrebno razmotriti mnoge razlike između Windows i Linuks platforme, posebno načine na koji su određeni programski koncepti, poput niti i semafora, implementirani.

Kroz poglavlje o evaluaciji ovog rešenja, uz pomoć primera koji obuhvata najvažnije delove, pokazano je da je FSM biblioteka uspešno prilagođena za rad na Linuks platformi.

Bitno je napomenuti da ovaj rad ne obuhvata NETfsm deo biblioteke. NETfsm je proširenje biblioteke čija uloga je da omogući njen izvršavanje preko mreže, kroz TCP ili UDP protokol. Razlog za ovo je nepotpuna implementacija ovog dela biblioteke, specifično dela o TPKT protokolu, koji služi za enkapsulaciju TCP protokola. Iz tih razloga se predlaže da TPKT implementacija, na način pogodan sa Linuks platformom, bude sledeće unapređenje ove biblioteke.

7. Literatura

- [1] M. Popović, Communication Protocol Engineering, CRC Publishing, 2006
- [2] I. Bašićević, M. Popović, I. Velikić “Use of Finite State Machine Based Framework in Implementation of Communication Protocols – A Case Study”, 2010
- [3] M. Pilipović, T. Srejić, M. Vučićević, I. Bašićević, M. Popović “Use of Qt Framework in University Courses on Development of Network Software“, Etran 2018
- [4] <https://www.ibm.com/developerworks/systems/library/es-win32linux-sem.html>, avgust 2020
- [5] <https://www.ibm.com/developerworks/systems/library/es-MigratingWin32toLinux.html> avgust 2020
- [6] Priručnik radnog okruženja za pisanje protokola, RTRK
- [7] <https://www.man7.org/linux/man-pages>, avgust 2020