УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ

Лука Билач

ПРЕДЛОГ РАСПОРЕЂИВАЧА ДОГАЂАЈА НА ПРИМЕРУ АНДРОИД СЕРВИСА ИНФОЗАБАВНОГ СИСТЕМА

МАСТЕР РАД

Нови Сад, 2023.

УНИВЕРЗИТЕТ У НОВОМ САДУ ● **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**
21000 НОВИ САД, Трг Доситеја Обрадовића 6

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, **РБР**:		
Идентификациони број, **ИБР**:		
Тип документације, **ТД**:		Монографска документација
Тип записа, **ТЗ**:		Текстуални штампани материјал
Врста рада, **ВР**:		Дипломски – мастер рад
Аутор, **АУ**:		**Лука Билач**
Ментор, **МН**:		**проф. др Илија Башичевић**
Наслов рада, **НР**:		**ПРЕДЛОГ РАСПОРЕЂИВАЧА ДОГАЂАЈА НА ПРИМЕРУ АНДРОИД СЕРВИСА ИНФОЗАБАВНОГ СИСТЕМА**
Језик публикације, **ЈП**:		Српски / латиница
Језик извода, **ЈИ**:		Српски
Земља публиковања, **ЗП**:		Република Србија
Уже географско подручје, **УГП**:		Војводина
Година, **ГО**:		**2023**
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)		
Научна област, **НО**:		Електротехника и рачунарство
Научна дисциплина, **НД**:		Рачунарска техника
Предметна одредница/Кључне речи, **ПО**:		**Распоређивање догађаја, Андроид оперативни систем, базен нити, међупроцесна комуникација**
УДК		
Чува се, **ЧУ**:		У библиотеци Факултета техничких наука, Нови Сад
Важна напомена, **ВН**:		
Извод, **ИЗ**:		**У овом мастер раду, истражени су изазови конкурентности у Андроид оперативном систему, који користи комбинацију вишенитног извршавања и слања догађаја како би подржао напредне задатке Андроид апликација. Посебно смо се фокусирали на проблем трка заснованих на догађају и недостатак синхронизације између одговарајућих повратних метода, што може довести до нежељених последица. Како бисмо превазишли ове изазове, предлажемо ефикасну имплементацију распоређивача догађаја, узимајући у обзир време доласка догађаја, приоритет клијента, броја необрађених догађаја и фреквенције доласка догађаја. Наш приступ има за циљ побољшање перформанси система, испуњење временских захтева и одржавање конзистентности тока података.**
Датум прихватања теме, **ДП**:		
Датум одбране, **ДО**:		
Чланови комисије, **КО**:	Председник:	**проф. др Мирослав Поповић**
	Члан:	**ванр. проф. др Драган Пејић** — Потпис ментора
	Члан, ментор:	**проф. др Илија Башичевић**

UNIVERSITY OF NOVI SAD ● **FACULTY OF TECHNICAL SCIENCES**
21000 NOVI SAD, Trg Dositeja Obradovića 6

KEY WORDS DOCUMENTATION

Accession number, **ANO**:			
Identification number, **INO**:			
Document type, **DT**:		Monographic publication	
Type of record, **TR**:		Textual printed material	
Contents code, **CC**:		Master Thesis	
Author, **AU**:		**Luka Bilač**	
Mentor, **MN**:		**Ilija Bašičević, PhD**	
Title, **TI**:		**Proposal of Event Scheduler of an Android In-Vehicle Infotainment Service**	
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**:		**23**	
Publisher, **PB**:		Author's reprint	
Publication place, **PP**:		Novi Sad, Dositeja Obradovica sq. 6	
Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes)			
Scientific field, **SF**:		Electrical Engineering	
Scientific discipline, **SD**:		Computer Engineering, Engineering of Computer Based Systems	
Subject/Key words, **S/KW**:		**Event scheduler, Android open source project, thread pool, interprocess communication**	
UC			
Holding data, **HD**:		The Library of Faculty of Technical Sciences, Novi Sad, Serbia	
Note, **N**:			
Abstract, **AB**:		**In this master's thesis ,we have explored the challenges of concurrency in the Android operating system, which utilizes a combination of multi-threaded execution and event dispatching to support advanced tasks in Android applications. We specifically focused on the problem of event-based race conditions and the lack of synchronization between corresponding callback methods, which can lead to undesirable consequences. To overcome these challenges, we propose an efficient implementation of an event scheduler, considering the arrival time of events, client priority, the number of pending events and event arrival frequency. Our approach aims to improve system performance, meet time requirements, and maintain data flow consistency.**	
Accepted by the Scientific Board on, **ASB**:			
Defended on, **DE**:			
Defended Board, **DB**:	President:	**Miroslav Popović, PhD**	
	Member:	**Dragan Pejić, PhD**	Menthor's sign
	Member, Mentor:	**Ilija Bašičević, PhD**	

Захвалност

За саветовање и посвећеност приликом израде завршног рада, захвалност дугујем свом ментору, проф. др Илији Башичевић, на Факултету техничких наука у Новом Саду.

Посебну захвалност дугујем комплетном Лава тиму на подршци и саветима током писања рада.

Желим да изразим дубоку захвалност својој породици, која ме је подржала током читавог овог пута, и мојој девојци Сањи што је била уз мене, пружала ми подршку и што се ангажовала и издвојила време за читање овог рада.

У Новом Саду, август 2023. године Лука Билач

САДРЖАЈ

Списак слика

Списак табела

Скраћенице

IEEE - *Institute of Electrical and Electronics Engineers*

ASHMEM – *Anonymous Shared Memory*

LMK – *Low Memory Killer*

RPC – *Remote Procedure Call*

HAL - *Hardware Abstraction Layer*

ART - *Android Runtime*

AOT – *Ahead Of Time*

JIT – *Just In Time*

CMS – *Concurrent Mark and Sweep*

SDK – *Software Development Kit*

PID – *Process identification*

ADB – *Android Debug Bridge*

AIDL – *Android Interface Definition Language*

HDR – *High Dynamic Range*

SNPE – *Snapdragon Neural Processing Engine*

EHM – *Event Handler Manager*

DSP – *Digital Signal Processor*

1. Увод

Као најпопуларнији оперативни систем на мобилним уређајима, Андроид користи комбинацију вишенитног извршавања и слање асинхроних догађаја како би подржао напредне задатке које извршавају Андроид апликације. Међутим, ово доводи до појаве сложених проблема конкурентности који су чести у Андроидовом систему [1]. Прослеђивач догађаја у Андроид систему омогућава оперативном систему да реагује на акције корисника и система путем генерисањем догађаја. Сваки догађај покреће одговарајућу повратну методу. Недостатак синхронизације између ових метода повратних позива [2], чији редослед извршавања није детерминистичан, доводи до трка заснованих на догађају.

У систему за рад у реалном времену, јако је битан концепт доделе приоритета одређених активности с циљем бољег распоређивања [3]. У моменту пристизања већег броја догађаја, онај са вишим приоритетом треба да има већу шансу извршавања у односу на догађај нижег приоритета. Један од већих проблема, са којима се сусрећу инжењери у овом пољу, јесте да обезбеђивање обрада свих догађаја и сигурност извршавања свих потребних акција у прихватљивом временском оквиру [4]. Да би се могла обезбедити сигурност потребно је имати неки вид распоређивања догађаја. Неке од битних разлога за постојање распоређивача догађаја су побољшање перформанси система, тако што се осигурава ефективна обрада догађаја у одређеном редоследу, да би се испунили временски захтеви, да се грешке превентивно одстране, и да се задржи конзистентност тока података. У овом раду, предлажемо једну имплементацију распоређивача догађаја, у зависности од времена доласка одређеног догађаја, приоритета клијента као и од фреквенције обраде и фреквенције доласка догађаја, на примеру вишенитног Андроид сервиса.

Програмско решење развијено је у склопу Лава пројекта. Лава нуди пример имплементације као и генератор, за успостављање комуникације између Андроид Инфо-забавног система и напредног система за помоћ при вожњи, на механизму сервисно оријентисане архитектуре, која се назива CommonAPI. Андроид страна је имплементирана и тестирана на Qualcomm SA8155P плочи [5].

Резултати датог решења су тестирани и проверени коришћењем Андроид апликација. Помоћу скупа испитних случајева утврђено је да се решење понаша идентично и исправно чиме је потврђена функционалност и исправност описаног решења. Део истраживања у оквиру предложеног решења је представљен на међународној IEEE конференцији у Опатији.

Рад је конципиран од седам поглавља.

Друго поглавље даје осврт на Андроид оперативни систем, са објашњењима реализације међупроцесне комуникације у Андроиду. Кратак опис архитектуре претплатника и издавача и објашњење алгоритама распоређивања догађаја.

Треће поглавље објашњава архитектуру решења уз појашњење основних компоненти дизајна решења.

Четврто поглавље садржи детаљан опис имплементације сервиса у Ц++ програмском језику, односно комплетну реализацију распоређивача догађаја, као и објашњење на који начин су распоређени догађаји, са факторима који утичу на једначину коефицијента приоритета.

У петом поглављу је описана верификација решења, са мерењем перформанси сервиса и приказана су ограничења са којима смо се сусрели.

Шесто поглавље садржи рекапитулацију урађеног у овом раду са даљим правцима у развоју.

У седмом поглављу је дат списак литературе које је била коришћена у овом раду.

2. Андроид екосистем

У овом поглављу су наведене теоријске основе које су потребне за разумевање овог рада. Теме су подељене на следећи начин :

- Софтверски слојеви Андроид оперативног система
- Андроид процеси
- Архитектура претплатника и издавача

2.1 Софтверски слојеви Андроид оперативног система

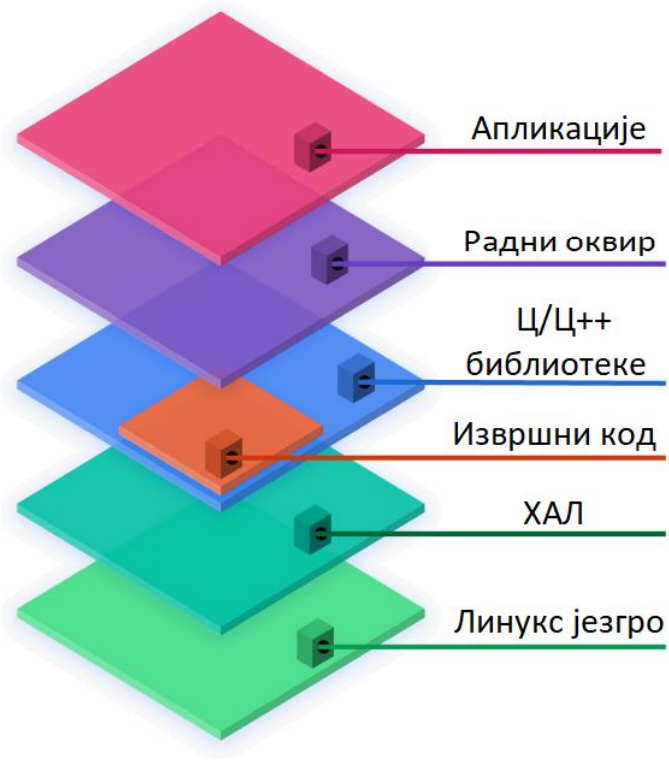
Андроид оперативни систем, под окриљем компаније Гугл, је првенствено био дизајниран за уређаје осетљиве на додир, као што су паметни телефони и таблети, и од тада је постао један од најпопуларнијих мобилних систем на свету, са бројним ажурирањима и новим верзијама које се објављују сваке године.

Андроид је један од најзаступљенијих оперативних система у мобилној индустрији, са преко 85% на тржишту према појединим изворима [6]. Андроид се наравно није зауставио само на мобилним уређајима, забележен је значајан пораст употребе Андроида као оперативног система и у наменским системима.

Једна од кључних предности Андроид оперативног система је што је његов код отвореног типа, што значи да је изворни код доступан програмерима да га модификују. Стога сви произвођачи, који користе Андроид као оперативни систем, имају потпуну слободу да га прилагоде својим потребама, независно од других произвођача. То је допринело широком прихватању Андроида од стране произвођача уређаја и програмера.

Доминантан програмски језик који се користи у развоју Андроид апликација је Јава. С обзиром да се Јава преводи у бајт-код, апликације се могу подједнако добро извршавати на свим платформама које су засноване на Андроиду, за разлику од апликација писаних у другим језицима, који се преводе у машински код намењен за одређену платформу.

Андроид је сачињен од софтверских слојева, који омогућавају ефикасно функционисање оперативног система и пружају подршку за различите функционалности [7]. За потребе лакшег разумевања поделићемо их на две целине, доњи део је претежно везан за хардвер, док је виши слој више везан за корисничке апликације (Слика 2.1 Архитектура Андроида).



Слика 2.1 Архитектура Андроида

Доњи слој чини Линукс језгро на који належе слој апстракције хардвера.

Андроид се темељи на Линуксовом језгру, што пружа основне функционалности оперативног система, укључујући управљање хардвером, управљање меморијом, управљање мрежом и сигурносне механизме. Он је одговоран за комуникацију између хардвера и софтвера на уређају. Андроид користи специјална проширења на Линукс језгру као што су Ashmem**,** повезивач (енг. Binder)**,** Логер (енг. Logger)**,** LMK**,** управљач напајања (енг. Power Manager)**,** Алат за отклањање грешака у језгру (енг. Kernel Debugger**)** [8]**.**

Ashmem је проширење које је додато на Линукс језгро специфично за Андроид платформу ради омогућавања ефикасног дељења меморије између различитих процеса на Андроид уређајима. Осмишљен је да буде ефикасан и да смањи утицај на перформансе система, коришћењем механизма страница за управљање меморијом. Такође, укључује механизме заштите како би се осигурало да само привилеговане апликације приступају дељеној меморији. Осмишљен је да буде компатибилан са постојећом Андроид спрегом за

програмирање апликација и механизмима за управљање меморијом, што представља једноставно коришћење функционалности Ashmem без великих изменама у коду.

Повезивач је механизам комуникације и размене података између процеса, специфичан за Андроид платформу. Омогућава комуникацију између процеса на Андроид уређајима. Имплементира међупроцесну комуникацију, као што су размена порука и RPC, омогућавајући сигурну и ефикасну комуникацију. Специфичан је по коришћењу система референци на објекте. Сваки објекат који се дели путем повезивача, има јединствен идентификатор који омогућава приступ објекту и коришћење његових података.

Логер проширење је део инфраструктуре за бележење догађаја и прикупљања дневника на Андроид уређајима. Омогућава праћење различитих догађаја и активности у језгру оперативног система. Има могућност прикупљања различитих врста записа као што су системски догађаји, грешке, упозорења, информације о перформансама и други корисни подаци. Ови записи су корисни приликом дијагностике проблема и анализе понашања уређаја.

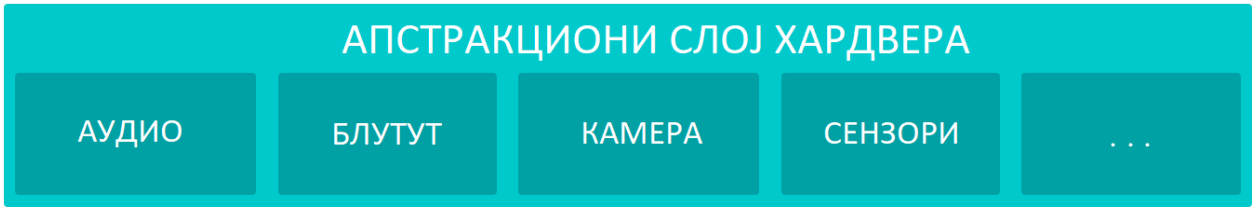
LMK је део Линукс језгра прилагођен за Андроид платформу, са задатком ефикасног управљања ограниченим ресурсима меморије на Андроид уређају. Када се систем приближи граници доступне меморије, има задатак да препозна и идентификује процесе који заузимају превише меморије, а нису активно коришћени. Користи алгоритме и хеуристике како би одлучио који процеси треба да буду терминирани, како би се ослободио простор у меморији. LMK додељује приоритете процесима на основу њихове важности и утицаја на систем. Процеси са нижим приоритетима имају већу вероватноћу терминирања приликом недостатака меморије, док процеси са вишим приоритетима имају већу вероватноћу да остану активни. Такође, има кључну улогу у одржавању стабилности Андроид система, спречавајући ситуације у којима недостатак меморије може довести до пада система или спорог одзива, обезбеђује равнотежу између ефикасног коришћења меморије и одржава перформансе система.

Управљач напајања има улогу у управљању потрошњом енергије на Андроид уређајима, тако што оптимизује рад уређаја како би смањио потрошњу батерије и продужио трајање исте. Врши контролу рада различитих уређаја и компоненти на уређају као што су процесор, екран, бежичне везе, сензори и други. Прилагођава њихову употребу и перформансе у складу са захтевима система и корисника. Нуди функционалности као што су прелазак уређаја у различите режиме мировања, искључивањем и укључивањем одређених активности док уређај није активан. Омогућава подешавање политике управљања енергијом, како би се прилагодио специфичним потребама корисника, такође је задужен и за приказ информација о потрошњи енергије.

Алат за отклањање грешака у језгру је алат уграђен у Линукс језгро за потребе Андроид платформе. Омогућава програмерима и инжењерима да анализирају и решавају проблеме у језгру оперативног система. Неке од функционалности које нуди су :

- Уклањање грешака језгра – интерактивно окружење за анализу и истраживање стања језгра током истраживања
- Интерактивна контрола – могућност прекидања извршавања језгра у одређеном тренутку
- Помоћ при оптимизацији – праћење перформансе, идентификација уског грла, анализирање времена одзива
- Интеграција са развојним окружењем – могућност интегрисања са различитим развојним окружењима

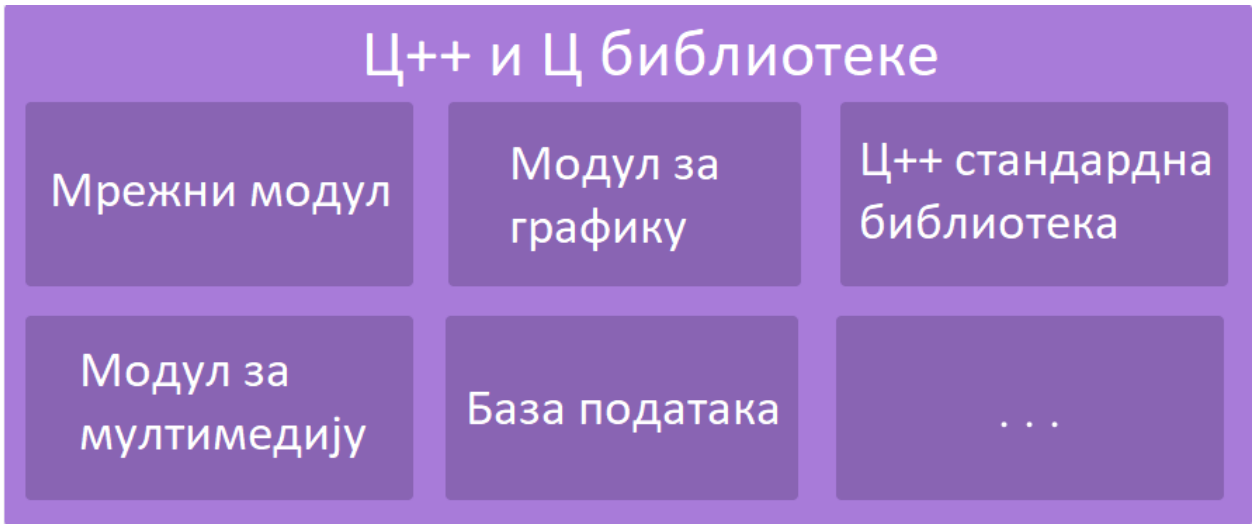
HAL, односно слој за апстракцију хардвера има кључну улогу у олакшавању интеракције између хардверских компоненти и виших слојева софтвера. Делује као посреднички слој, омогућавајући хардверски независном софтверу да комуницира са основним хардверских компонентама Андроид уређаја [8]. Сачињен је од скупа библиотека и драјвера који пружају стандардизовану спрегу за софтверске компоненте, омогућавајући им комуникацију са специфичним функционалностима хардвера (Слика 2.2). Апстрахује комплексне детаље хардверских имплементација, чиме штити горње софтверске слојеве од сложености уређаја и омогућава лакшу преносивост на различите хардверске платформе. Модуларни приступ Андроида омогућава додавање HAL модула и проширења, чиме је оспособљена подршка за нове хардверске функције или унапређења без захтева за модификацијама целокупног софтверског система. HAL доприноси укупној безбедности и приватности Андроид система спровођењем контрола приступа и дозвола за хардверске ресурсе. Осигурава да осетљиве функционалности буду адекватно изоловане и заштићене. Тренутна истраживања и напори усмерени су на еволуцију HAL за потребе подржавања нових хардверских технологија, попут проширене стварности, виртуелне реалности и напредних сензора. Ова еволуција има за циљ пружање развојним тимовима унапређене могућности и побољшање корисничког искуства.



Слика 2.2 Апстркациони слој хардвера

Виши слој је највише од користи програмерима апликација. Њега чине слој Ц++ и Ц библиотека, ART, слој развојног оквира и апликативни слој [8].

Андроид поседује разне библиотеке које пружају бројне функционалности и ресурсе за развој апликација. Ове библиотеке укључују различите модуле за графику, мултимедију, мрежно програмирање, базу података, сигурност, управљање енергијом и још много тога. Неке од кључних библиотека укључених у Андроид су мрежни модул, модул за графику, Ц++ стандардна библиотека, модул за мултимедију, база податак итд (Слика 2.3).



Слика 2.3 Слој Ц++ и Ц библиотека

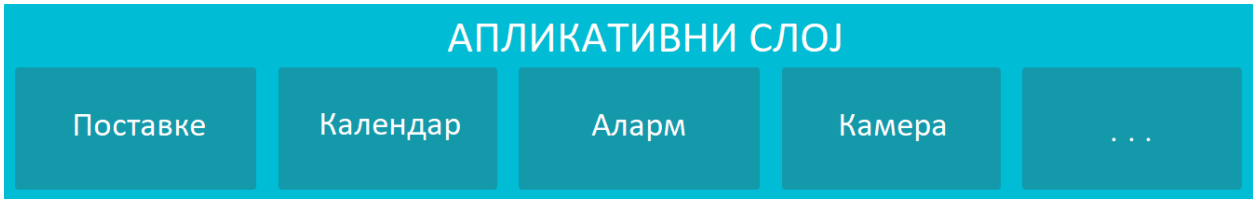
ART је окружење за извршавање и управљање апликацијама на Андроид платформи. Почевши од Андроид 5.0 ART је заменио старију Далвик виртуелну машину. Донео је неколико побољшања, укључујући превођење кода пре извршавања AOT, побољшане перформансе и унапређено управљање меморијом. Са AOT-ом, бајт код апликације се преводи у машински код током фазе инсталације апликације. То омогућава брже извршавање апликација, из разлога што нема потребе за JIT превођењем током извршавања. Претходна JIT превођења у Далвик виртуелној машини стварала су одређени процесни трошак. ART смањује тај трошак, резултирајући бржим покретањем апликације и побољшаном укупном перформансом. ART је такође увео ефикаснији механизам за управљање меморијом звани CMS. Сакупљач смећа ради паралелно са апликацијом, смањујући застоје и побољшавајући одзив система. Уз то проширење, увео је и функционалност сажимања сакупљача смећа (енг. Garbage Collection compacting) која реорганизује меморијски простор како би се смањила фрагментација меморије.

Слој радног оквира пружа програмерима алате, спрегу за програмирање апликација и библиотеке за изградњу Андроид апликација. То је слој апстракције између системских компоненти нижег нивоа и слоја апликација, нуди скуп поновно употребљивих компоненти

и сервиса који поједностављују процес развоја и обезбеђују доследно понашање на различитим уређајима. Језик који је најчешће коришћен за имплементацију је Јава програмски језик. Неке од кључних компоненти које спадају у слој развојног оквира су услужиоци попут, Услужиоца активности, локација, пакета, обавештења, ресурса, телефоније, оквир, Добављач садржаја, Систем за приказ, итд (Слика 2.4). Најважнији део овог слоја је Андроид SDK, који садржи све неопходне алате за развој апликација, укључујући емулатор, развојне алате, документацију и пример кодова. Такође, Андроидом слој радног оквира укључује модуле за управљање корисничком спрегом, управљање ресурсима, подршку за мрежно повезивање, приступ бази података и још много тога.

Слика 2.4 Слој радног оквира

На самом врху се налази апликативни слој Андроида. Омогућава израду разноврсних апликација, укључујући игре, продуктивне алате, комуникацијске апликације, мултимедијалне апликације и још много тога (Слика 2.5). Такође овде се налазе апликације које су пре инсталиране на уређају, као што су календар, контакти, прегледач, поштански клијент, музички плејер и друге.



Слика 2.5 Апликативни слој Андроида

2.2 Андроид Процеси

Процеси у Андроиду су основне компоненте Андроид оперативног система. Процес представља инстанцу покренуте апликације и одговоран је за извршавање кода и

управљање ресурсима повезаним са том апликацијом. Андроид користи систем приоритета и управљања меморијом како би ефикасно руковао процесима [9].

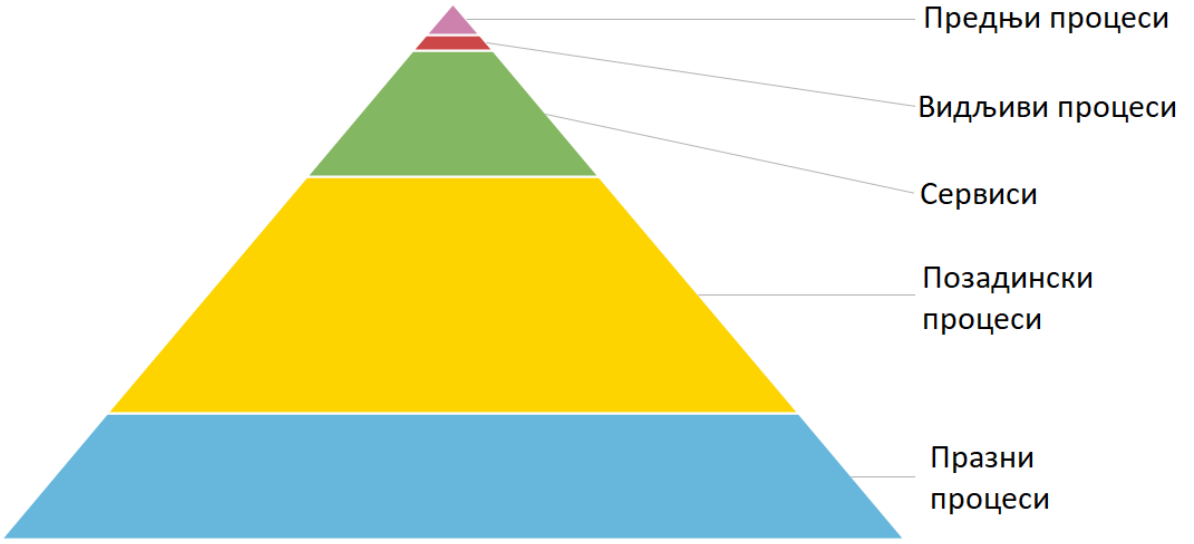
Слично, као и код Линукс процеса, сваки Андроид процес поседује јединствен идентификатор, који се назива PID, за потребе идентификације и руковања процесима. Андроид прати модел вишепрограмског рада, односно модел у коме више различитих процеса имају могућност извршавања у исто време, пружајући кориснику истовремену интеракцију са различитим апликацијама.

Андроид процеси се могу класификовати у две основне групе, процеси који раде у првом плану и процеси које раде у позадини [10]. Предњи процеси, односно процесу који су у првом плану су везани за активности и апликације које су тренутно видљиве кориснику. Такви процеси поседује већи приоритет и снабдевају се са више ресурса, као што су процесорско време и меморија, са циљем бољег корисничког искуства. Поред апликација које су видљиве кориснику, у ову групу процеса спадају и процеси који очекује интеракцију са корисником (Слика 2.6).

На другој страни, позадински процеси су везани за апликације које нису тренутно видљиве кориснику. Такви процеси, у уобичајеном случају, имају мањи приоритет и мање ресурса им је додељено на коришћење у поређењу са процесима првог плана. Андроид дозвољава процесима извршавање у позадини, међутим они могу бити разлог агресивније контроле над ресурсима, са циљем оптимизације перформанси система и животног века батерије. Уобичајено, позадински процеси извршавају послове као што су руковање мрежним захтевима, обрада података у позадини или ажурирање информација на основу системских догађаја.

Андроид такође уводи системске процесе који су одговорни за разне задатке на системском нивоу. Ови процеси рукују критичним функцијама, као што је руковање системским сервисима, контрола улазно излазних операција, обезбеђивање стабилности и безбедности система. Поседује уникатан процесни идентификатор и непрестано се извршавају у позадини као потпора свим операцијама на Андроид уређају.

Системски администратори, као и Андроид програмери имају доступне алате и спреге за програмирање којима могу ефективно руковати процесима. Андроид радни оквир нуди механизме за покретање, заустављање и контролу процеса на основу корисничке интеракције и системских захтева. Додатно, алат као што је ADB нуди детаљан увид о перформансама процеса и коришћеним ресурсима.



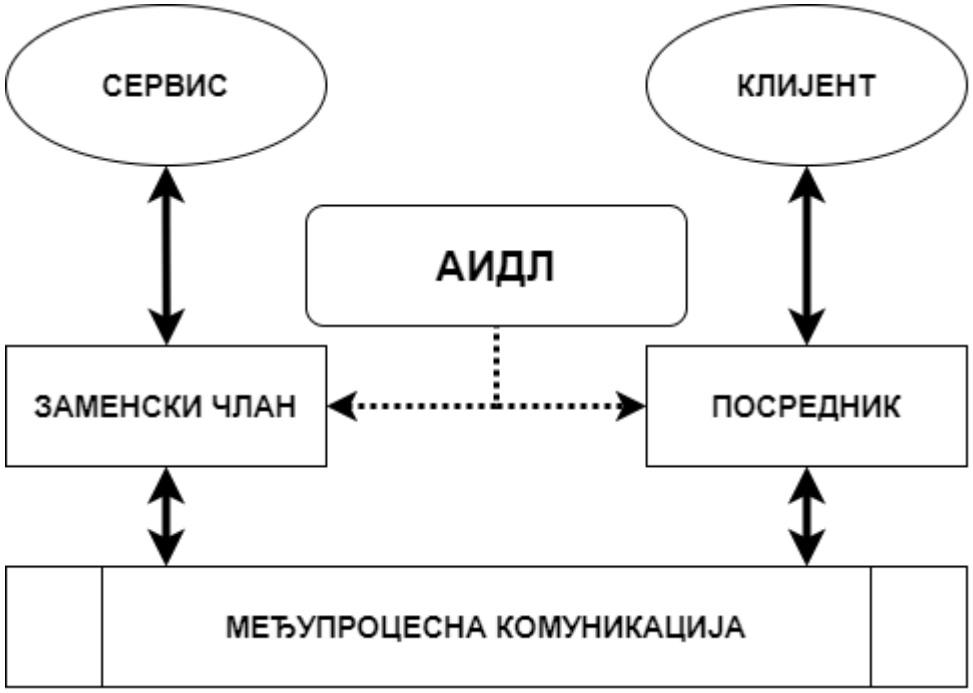
Слика 2.6 Пирамида приоритета процеса

Као и код Линукс процеса, Андроид процеси поседује толерантност (енг. Niceness) вредност, односно механизам коришћен за расподелу приоритета процеса и алокацију системских ресурса на одговарајући начин. Толерантност представља ниво приоритета процеса, такав да што је мањи број, то је већи приоритет. Распон којим се креће ова вредност је од негативних двадесет, односно највећи приоритет, па до позитивних деветнаест, односно најмањи приоритет. Контролом ове вредности, програмер поседује могућност утицаја на расподелу процесорског времена између процеса. Мања вредност дозвољава процесу да добије више процесорског времена, док већа вредност утиче на смањење истог. Овај механизам расподеле приоритета обезбеђује поштену алокацију ресурса и дозвољава да критичан процес добије више процесорског времена на располагање, док мање критични или позадински процеси функционишу са мање процесорског времена.

2.2.1 AIDL

У Андроид окружењу, сваки процес поседује свој адресни простор, што значи да један процес не може приступити подацима другог процеса [11]. Из тог разлога, дефинисан је механизам међупроцесне комуникације, који изолује податке и спречава штетну директну интеракцију процеса. Да би клијент могао да комуницира са сервисом посредством међупроцесне комуникације, неопходно је дефинисати програмску спрегу, која се зове AIDL. Он обезбеђује синхрон вид комуникације између процеса. Покретањем Андроид генератора над AIDL фајлом, добијамо сервисну и клијентску инстанцу, односно заменски члан (енг. *Stub*) и посредник (енг. *Proxy*) класу [12]. Посредника користи клијент, да би имао приступ сервису преко RPC. Са друге стране сервис користи заменски члан, за потребе имплементације удаљених метода (Слика 2.7). Поред те улоге, учествује и у слању и

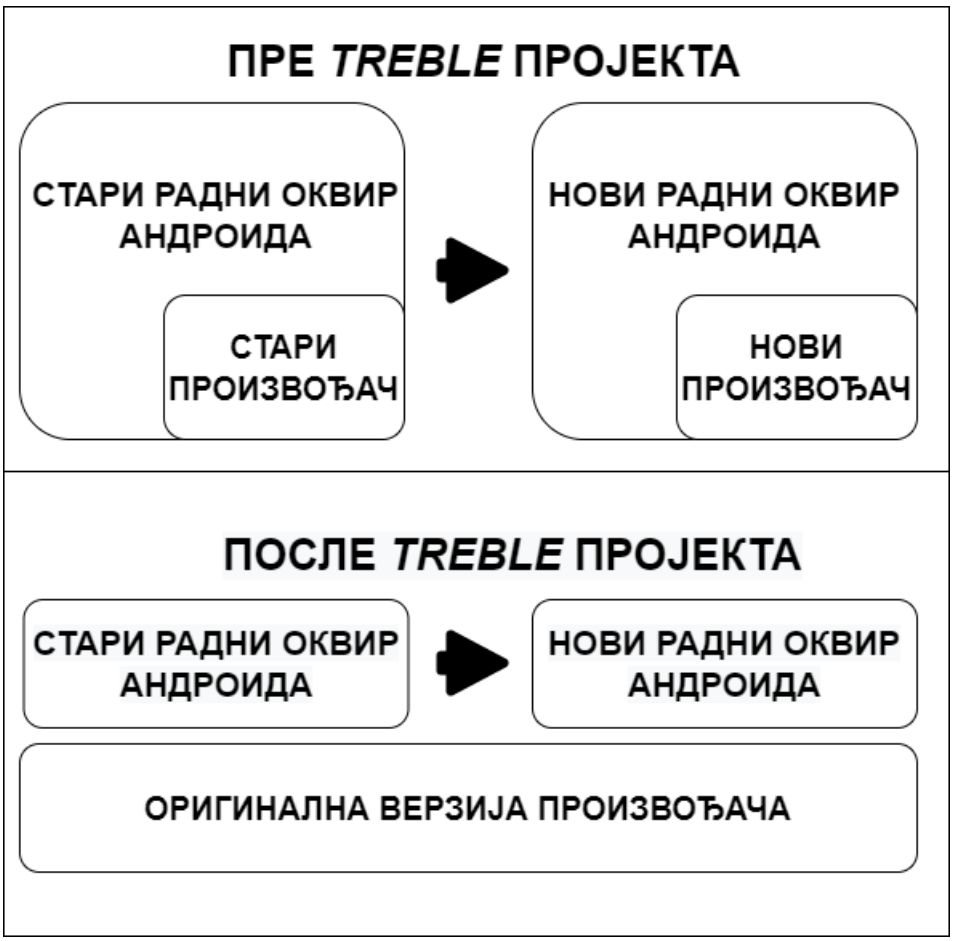
примању података сервиса. Да би се идентификовала удаљена метода, неопходан је објекат повезивача са стране сервиса. Он потиче од драјвера повезивача, који обезбеђује удаљени позив процедуре.



Слика 2.7 Међупроцесна комуникација помоћу AIDL спреге

2.2.2 Повезивач

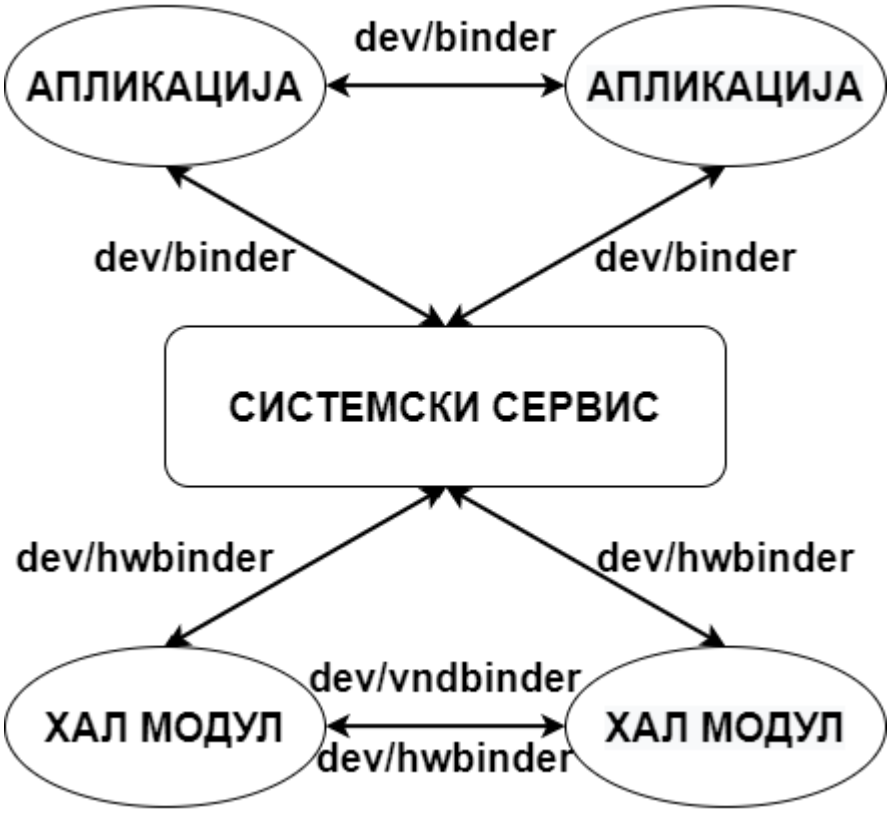
До Андроида 8 постојао је само један тип драјвера повезивача, међутим у оквиру *Treble* пројекта дошло је до промене [13]. Наиме, *Treble* пројекат представља амбициозни вид реорганизације архитектуре Андроида, уводећи поделу између радног оквира и нижег слоја софтвера који је највише везан за хардвер, који зовемо произвођач (eng. *Vendor*). Раније, пре него што је спрега произвођача уведена, велика количина кода је морала бити обновљена и надограђена приликом преласка на новију верзију Андроида. Међутим, са стабилном спрегом, приликом преласка на нову верзију Андроида, није неопходно надоградити сав код, већ само радни оквир док имплементација произвођача остаје иста (Слика 2.8). То је обезбеђено остваривањем компатибилности радног оквира са ранијим верзијама произвођача. Из приложеног, закључујемо да је произвођачима уређаја дата већа флексибилност приликом унапређивања софтвера, која се одликује већом брзином, а мањом потрошњом ресурса



Слика 2.8 Произвођачки слој и *Treble* пројекат

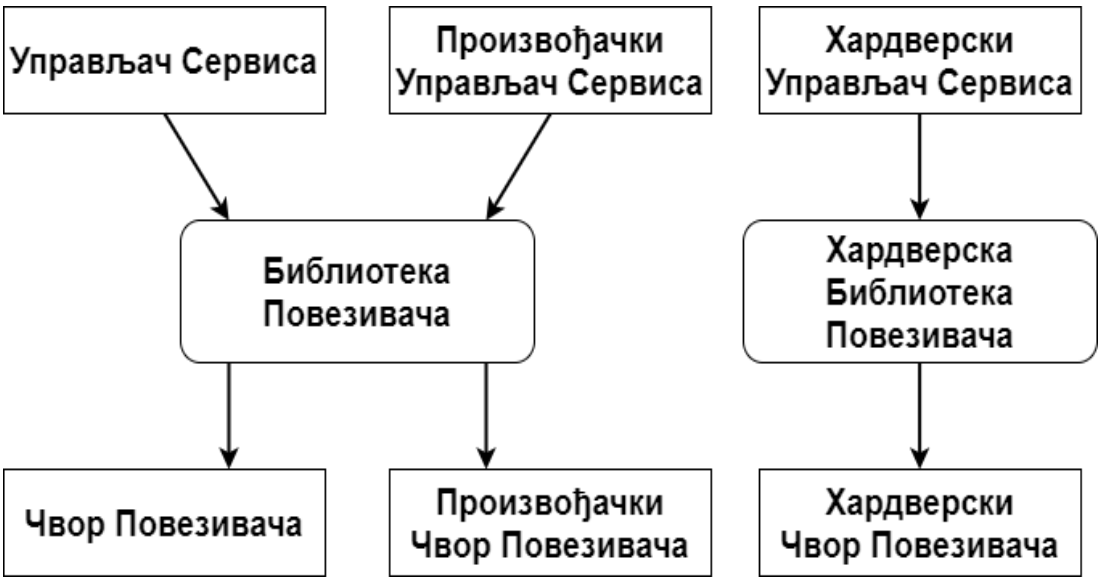
Пошто од Андроида 8 имамо два слоја која знатно повећавају коришћење повезивача, радни оквир и произвођач, дошло је до побољшања повезивача да би комуникација остала брза. Да би се јасно раздвојила комуникација уведен је контекст повезивача [14]. Сваки повезивач има свој чвор и свог управљача сервиса, коме се може приступити само преко одговарајућег чвора. Такође, ако се шаље чвор повезивача, прималац га мора примити коришћењем истог управљача коме припада тај чвор.

Тренутно, имамо три врсте чвора повезивача, */dev/binder*, */dev/hwbinder* и */dev/vndbinder* [15]. Првом је додељена улога у комуникацији искључиво између процеса из радног оквира, други има улогу у комбинацији комуникација радног оквира и произвођача, док је трећи потпуно нови чвор, који постоји од Андроида 8, и задужен је за комуникација између произвођачких процеса (Слика 2.9).



Слика 2.9 Врсте повезивача и њихова улога

Да би сервис у Андроиду био на располагању клијентима, неопходно је да се он региструје преко јединственог управљача сервиса, који ће се надаље старати о позивима клијента [16]. Поред поделе у чворовима повезивача, дошло је и до поделе у управљачима сервиса, те такође имамо регуларни управљач сервисима који је сада задужен за радни оквир и апликативне процесе, као и произвођачки управљач сервисима, који је сада задужен само за произвођачке сервисе. Иако је овом поделом покривена комуникација свих слојева, битно је напоменути да није могуће комбиновати */dev/binder* и */dev/vndbinder* чвор (Слика 2.10), то је лимитација софтверског развојног алат SDK, повезивача, а такође и договор Андроид заједнице.



Слика 2.10 Библиотеке повезивача

2.2.3 Интенти и Дељена меморија

Поред Андридовог повезивача постоје још два вида међупроцесне комуникације, а то су интент и Ashmem.

Интент дефинише асинхрон вид комуникације који покреће сервис [17], неку активност, или пријемнике за неусмерене поруке (енг. *Broadcast Receivers*). Ако се интент користи при побуди одређене активности унутар једне исте апликације, онда је реч о експлицитном интенту, за разлику од имплицитног интента који се користи као побуда сервиса или неке активности изван оригиналне апликације. Поред побуде, интент има још једну особину, може да проследи групу података која се назива пакет (енг. *Bundle*), заједно са побудом.

Други вид међупроцесне комуникације представља анонимна дељена меморија. У поређењу са Линукс дељеном меморијом, ако неки процес захтева да се ослободи меморија за његове потребе, Андроид оперативни систем ће је ослободити.

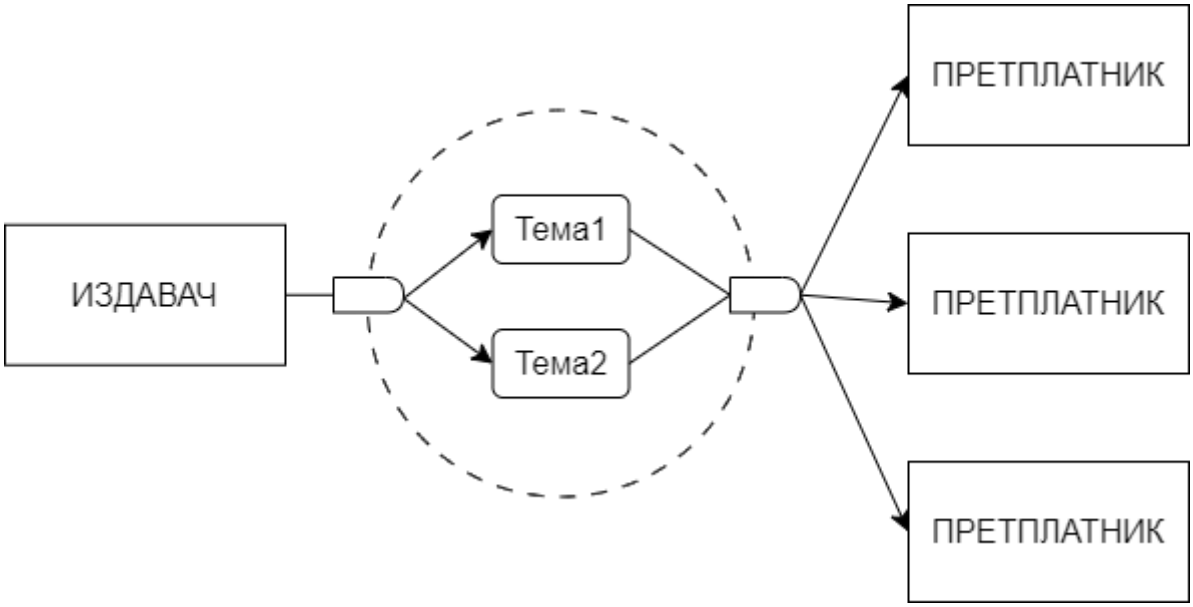
2.3 Архитектура претплатника и издавача

Механизам претплатника и издавача је чест начин имплементације програмирања заснованог на догађајима у Ц++ програмском језику. Овај механизам омогућава комуникацију између различитих делова програма путем размена порука [3].

У овом механизму, издавачи су одговорни за слање порука или догађаја, док су претплатници одговорни за примање и обраду истих. Комуникација се обично обавља путем реда порука или система за претплату и издавање.

Неки од кључних корака представљају, дефинисање порука које ће бити размењене и на који начин, затим регистрација претплатника, односно начин на који ће претплатник бити обавештен када за то буде било потребе, и на крају само слање и обраде исте поруке (Слика 2.11).

Овај механизам представља флексибилан и скалабилан начин програмирања догађаја, омогућавајући различитим деловима програма да комуницирају међусобно, док је сама имплементација претплатника, односно издавача одвојена, што може учинити програм лакшим за разумевање и одржавање [18]. У Андроиду, овај механизам се широко користи путем међупроцесне комуникације.



Слика 2.11 Мехаинзам издавача и претплатника

2.3.1 Алгоритми сортирања догађаја

Комуникациони механизам представља различите начине на које софтверске компоненте комуницирају међусобно, као што су методе, слање порука и догађаји.

Методе су уобичајени начин комуникације у објектно-оријентисаном програмирању, методе су функције повезане са објектом и могу бити позване од стране других објеката приликом захтевања извршавања одређене акција. Након позивања методе, обично се враћа вредност акције или се модификује стање објекта.

Слање порука је други механизам у коме компоненте једна другој шаљу поруке путем посредника или посредничког софтвера. Порука садржи информације о пошиљаоцу, примаоцу, и садржај поруке, а посредник усмерава поруку одговарајућом примаоцу. Слање порука се често користи у дистрибуираним системима, где компоненте могу радити на различитим машинама или у различитим процесима.

Догађаји су трећи вид комуникације, обавештавају да се нешто десило у омогућава другим компонентама да реагују на тај догађај, на пример клик дугмета на корисничкој спрези може покренути догађај, који се може обрадити од стране других компоненти у раду.

Планирање догађаја је процес организовања догађаја, да се обраде у одређено време или у одређеним интервалима. То се може постићи коришћењем мерача времена или планирањем догађаја који ће се активирати на основу одређених услова, као што је завршетак задатка или појава одређеног догађаја. Један уобичајен пример планирања догађаја је у контексту оперативних система, где оперативни систем распоређује процесе да се покрећу у одређено време или као одговор на одређене догађаје. Општи циљ планирања догађаја је управљање током догађаја у систему и осигуравање да се догађаји изврше у правилном редоследу, што је кључно за одржавање перформанси система.

Постоје неколико начина како извршити одређивање приоритета догађаја :

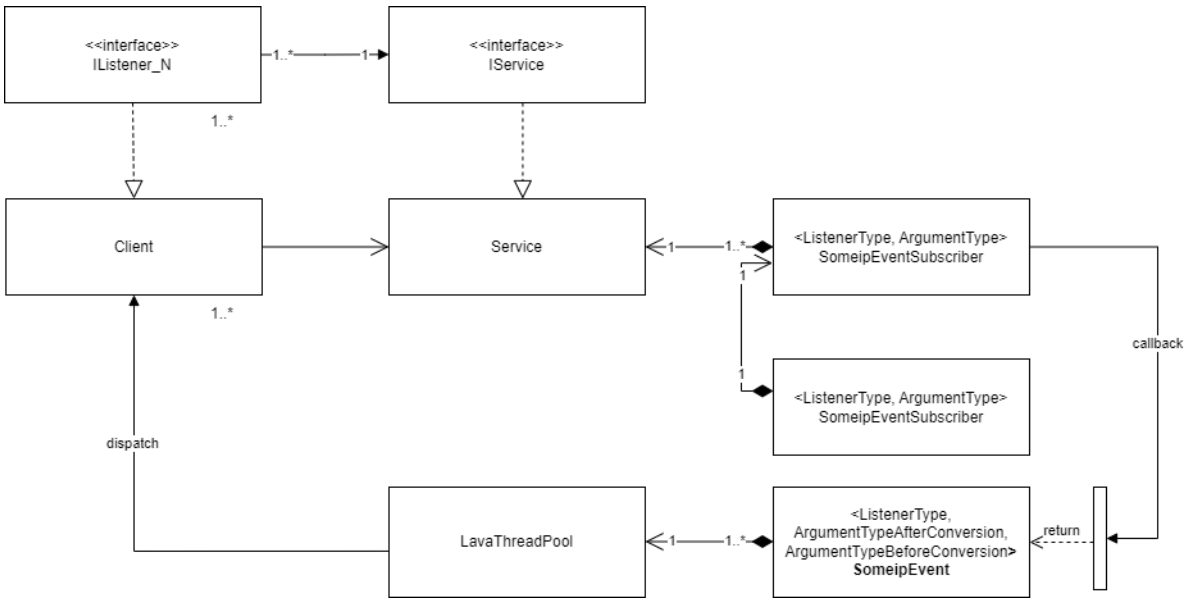
- Распоређивање по приоритетима – задацима се задаје одређени ниво приоритета на основу кога се одређује редослед извршавања. Распоређивач увек преузима задатак са највећим приоритетом.
- Кружно распоређивање – задацима се задаје временски оквир, распоређивач пролази кроз задатке и сваком даје шансу да се изврши у временском оквиру, пре него што пређе на следећи
- Распоређивање према најранијем року – уз задатак је везан крајњи моменат у времену, када задатак мора бити извршен. Задатак који се бира за извршавање ће бити онај са најранијим очекиваним временом извршавања
- Распоређивање према монотоној стопи – сваки задатак поседује очекивано време извршавања, а распоређивач бира онај задатак који има најкраће време извршавања
- Кружно распоређивање са тежинским фактором – слично попут кружног распоређивања, међутим сваки задатак поседује и тежински фактор, а распоређивач даје више времена задацима са јачим фактором. Овај вид распоређивања је користан када постоји подела у задацима, у којима су неки битнији од осталих
- Превентивно распоређивање – задаци поседује ниво приоритета, разликује се од распоређивању по приоритету по томе што распоређивач може да прекине обраду задатка мањег приоритета, за потребе извршавања задатка већег приоритета. Користи се у система за рад у реалном времену, где су заступљени критични задаци и чије кашњење може утицати на цео систем.

- Хибридно распоређивање – представља вид распоређивања у коме се комбинују више различитих начина распоређивања, као на пример приоритетно и кружно. Користи се у системима где различити задаци имају различите захтеве и ограничења
- Распоређивање са најмање преосталим временом – задацима се задаје временски моменат у коме треба да се заврше, такође и време извршавања. Распоређивач бира следећи задатак по принципу најмањег преосталог времена до краја извршавања, корисно је у системима за рад у реалном времену
- Распоређивање на основу ресурса – задацима се задаје приоритет на основу ресурса који су им потребни за извршавање, као што су меморија, процесорско заузеће, ширина тока податка на мрежи, итд. Задаци који користе више ресурса су посматрани као приоритетнији. Овај вид распоређивања ја користан у системима са ограниченим ресурсима
- Редови распоређивање са више приоритета – задацима су додељени различити нивои приоритета, распоређивач користи посебан ред за сваки ниво приоритета у који се смештају задаци. Редови већег приоритета се извршавају пре радова са нижим приоритетом
- Саморегулишуће распоређивање – задацима се додељују динамички приоритети, који се базирају на њиховој историји извршавања и конзумацији ресурса. Распоређивач модификује приоритет сваког задатка, са циљем да задаци који се чешће извршавају добију већи приоритет
- Вишекритеријумска додела приоритета – задаци су сортирани на основу више критеријума, као што су крајње време извршавања, битност задатка, конзумација ресурса и информација од корисника.
- Прилагодљива додела приоритета – задацима се задаје приоритет на основу тренутног стања и обиму посла система.
- Додела приоритета на основу машинског учења – задаци су сортирани помоћу алгоритама машинског учења, који узимају различите факторе приликом сортирања, као што су претходне перформансе, коришћење ресурса, понашање корисника. Корисно је у системима где су карактеристике и захтеви задатка динамички и тешки да се предвиде
- Додела приоритета на основу корисника – задаци се задаје приоритет на основу информација од стране корисника, као што су захтеви корисника и приоритети које корисник одреди. Корисно је у система где корисник има директан утицај на извршавање задатака.

3. Концепт решења

Ово поглавље описује реализацију и концепт распоређивача догађаја на примеру вишенитног Андроид сервиса написаног у Ц++ језику. Улога распоређивача јесте да рачуна коефицијент приоритета за сваки пристигли догађај на основу четири фактора, да сортира адекватно догађаје, чиме добијамо боље перформансе система као и контролисано извршавање и прозивање одговарајућих клијената у одговарајућем временском року.

Сам концепт решења се ослања на комуникацију посредством AIDL спреге. За почетак имамо AIDL спрегу која описује методе за иницијализацију комуникације између клијента и сервиса, као што је метода за претплату на сервис, чиме сервис добавља клијентску спрегу. Друга AIDL спрега служи за прозивање клијента од стране сервиса. Сервис ће добити спрегу посредством метода из првог AIDL, затим ће посредством друге AIDL спреге прозвати методу над жељеном спрегом одговарајућег клијента (Слика 3.1).



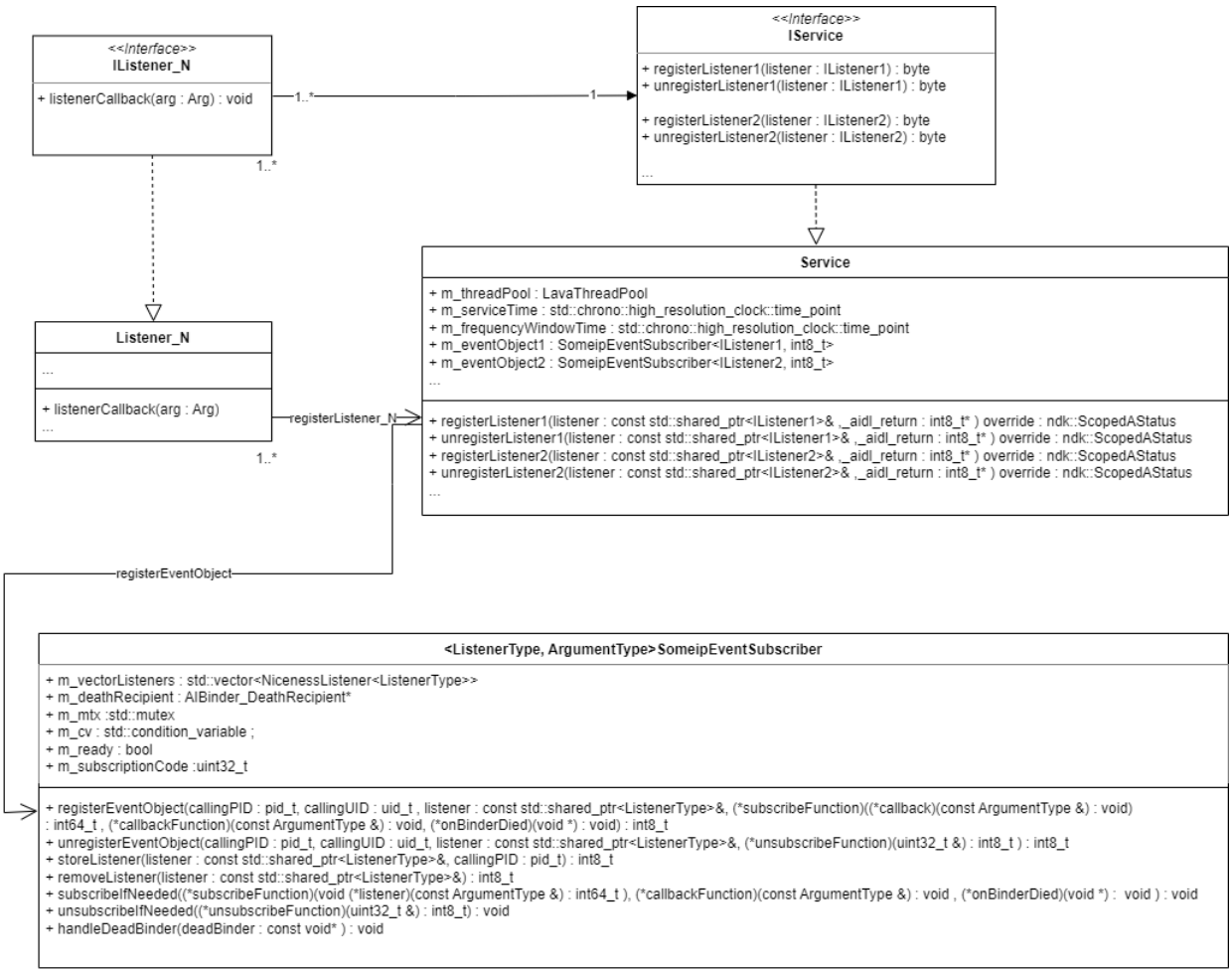
Слика 3.1 Архитектура решења

Да би се комуникација боље разумела можемо поделити само функционалност на два дела. Први представља сам концепт чувања клијената, с обзиром да је комуникација помоћу AIDL спреге увек једносмерна, комуникација коју иницира клијент ка сервису. Други концепт који ћемо објаснити је комуникација од стране сервиса ка клијенту.

3.1 Чување клијената

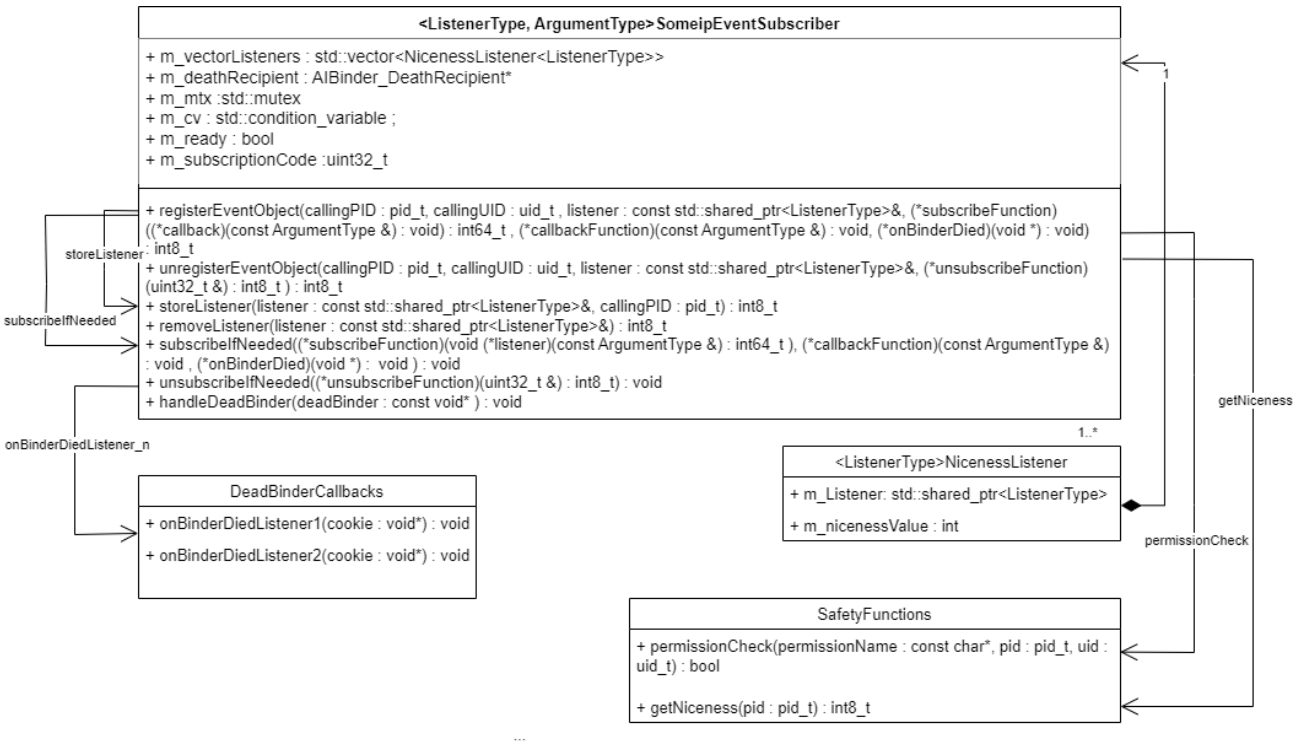
Почетак интеракције започиње клијент, тако што ће креирати инстанцу спреге која ће бити коришћена за прозивање односно механизам повратног позива. Спрега је дефинисан од стране сервиса, дефинишући повратне вредности као и аргументе који се пропагирају, односно шаљу од стране сервиса.

Након креирање објекта спреге за комуникацију, клијент прозива методу имплементирану на сервис страни посредством AIDL спреге, односно позива претплата на одговарајући догађај, и као аргумент прослеђује објекат спреге (Слика 3.2).



Слика 3.2 Архитектура чувања клијента

Након позива претплате на сервис, врши се провера да ли клијент има одговарајуће дозволе да прими тражени податак, за случај да нема одговарајуће дозволе, позив ће бити одбијен уз информацију о разлогу одбијања. Затим се проверава да ли је тај објекат већ регистрован, ако све те провере прођу, праве се одговарајући објекти и повратне методе за регистровање терминираног клијената, имплементирано од стране повезивача. Након тога се клијент додаје у контејнер клијената за одговарајући догађај и чека на прозивање од стране сервиса. У истом моменту приликом чувања клијената, преузима се коефицијент толерантности од процеса који зове регистровање на сервис, и у склопу објекта спреге се додаје у мапу. У случају да процес који је проследио објекат спреге заврши своје извршавање, доћи ће до прекида комуникације и клијент ће бити правовремено избачен из контејнера у којима се чувају клијенти (Слика 3.3).



Слика 3.3 Архитектура провера права клијента

3.2 Прозивање клијената

Други део концепта представља прозивање клијената од стране сервиса. Да би се обезбедила бржа интеракција са клијентима, користи се базен са нитима који ће прозивати клијенте паралелно у одговарајућем редоследу.

При доласку одређеног догађаја, прозива се сервис који пролази кроз све претплаћене клијенте за тај догађај. Над сваким објектом спреге клијента се прави нови објекат који у себи садржи објекат спреге и коефицијенте који ће бити коришћени за сортирање.

Коефицијент приоритета се рачуна на основу четири параметара.

Први параметар поседује највећи утицај на коефицијент приоритета, представља време доласка догађаја. У моменту доласка догађаја, преузима се тренутно време и рачуна се апсолутно време доласка догађаја.

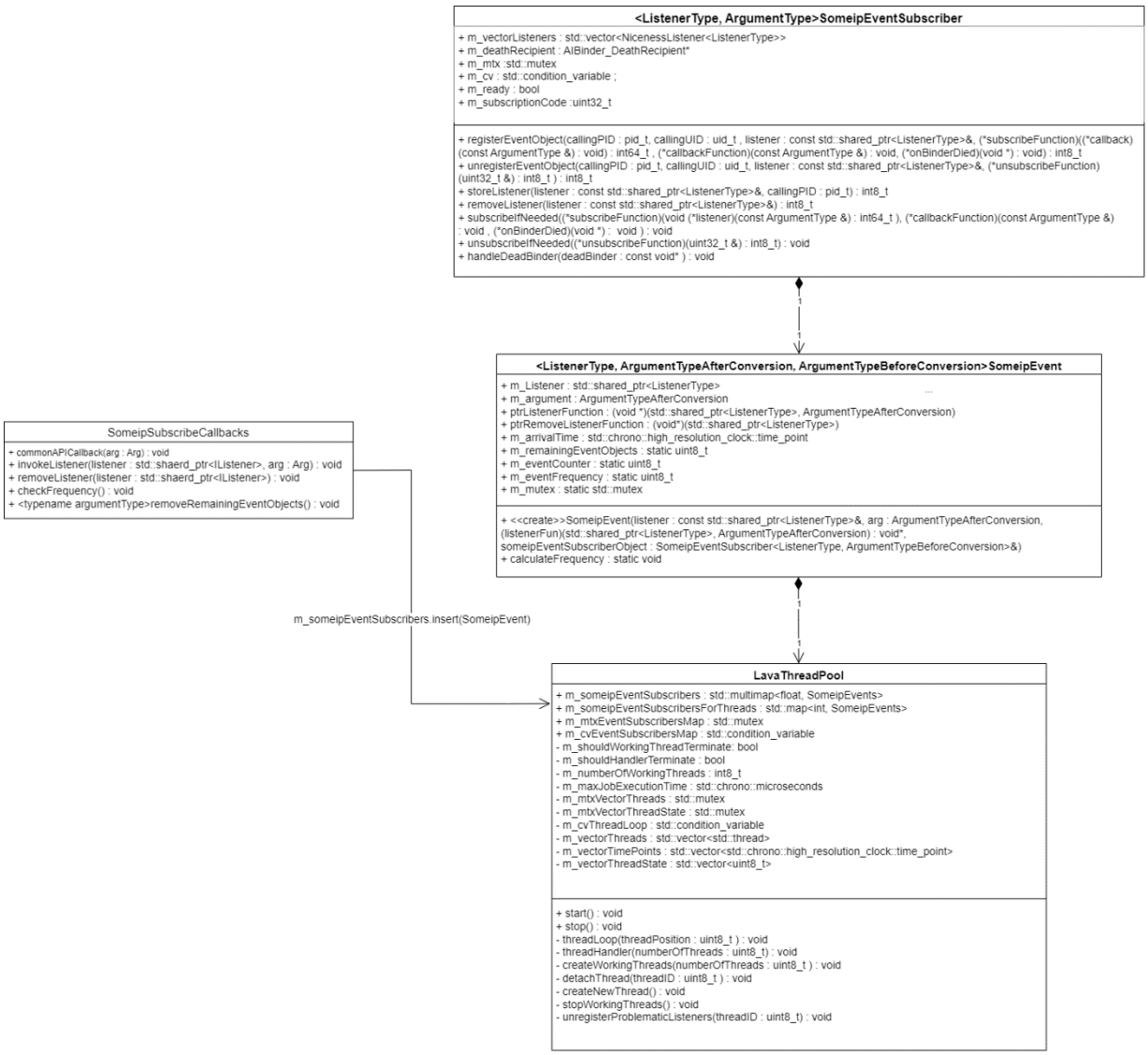
Други фактор чини толерантност која се преузима од идентификатора процеса. Толерантност је вредност која се поставља од стране програмера приликом покретања клијентске апликације. Распоређивач може да испуни захтеве програмера, чиме програмер има могућност утицаја на доделу приоритета догађајима.. Та вредност се креће од минус деветнаест до двадесет, негативнија вредност представља већи приоритет.

Трећи фактор представља број необрађених објеката догађаја, по догађају. Сваки пут када догађај пристигне, и направимо објекат догађаја по претплаћеном клијенту, увећава се бројач необрађених објеката догађаја. Када се објекат догађаја обради, и клијент буде

обавештени о пристиглом догађају, бројач се умањује. На основу тога имамо информацију о броју и стању необрађених клијената по догађају. Када следећи пут пристигне догађај, врши се провера необрађених објеката. У случају да постоје клијенти који нису обавештени о претходном догађају, а следећи догађај је стигао, објекти клијената који нису прозвани, ће бити избачени из реда чекања, и приоритет тог типа догађаја ће се увећати. Овим фактором имамо контролу над односом брзине пристизања догађаја и брзине прозивања клијената претплаћених на тај догађај. Битност овог фактора се огледа у томе, што у система за рад у реалном времену, можемо да утичемо на то, да не дође до догађаја која нису обрађени, што може нарушити рад клијентске апликације.

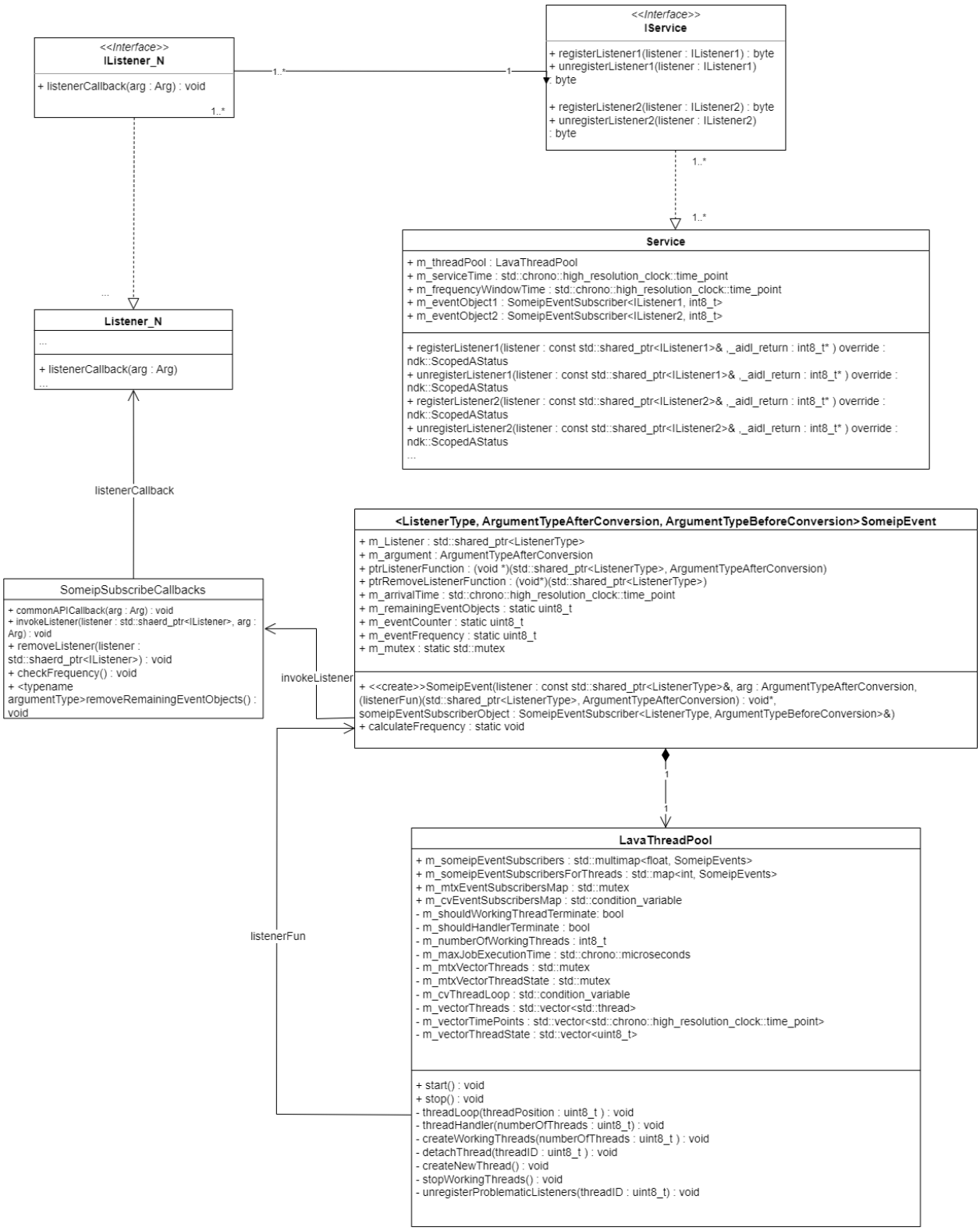
Четврти фактор је учесталост пристизања догађаја. Сваки пут када догађај пристигне вршимо проверу учесталости долазака догађаја. Поседујемо временски оквир сервиса, када пристигне догађај проверавамо да ли је истекао временски оквир сервиса, ако јесте, преузимамо тренутни бројач догађаја од тренутка последњег освежавања учесталости, након чега се бројач враћа на почетно стање и померамо временски оквир сервиса. Број пристиглих догађаја се множи коефицијентом који се рачуна од величине толерантности. Четвртим фактором покривен је случај у коме догађају који чешће пристижу, добију мало већи приоритет од осталих догађаја, без нарушавања толерантност фактора, чиме клијент има директан утицај на коефицијент приоритета.

Сви фактори се сабирају, што је коефицијент мањи, то му је већи приоритет, односно што је пре пристигао пре ће се и обрадити да би се задовољила главна особина рада система у реалном времену. Када се заврши рачунање коефицијената, у мулти мапу се додаје објекат, чији је кључ, управо добијен коефицијент, а вредност представља објекат догађаја, креиран на основу сваког претплаћеног клијента (Слика 3.4).



Слика 3.4 Архитектура прозивања клијента први део

Након што су израчунати коефицијенти и додати објекти за пристигли догађај, базен нити почиње за прозивањем клијената. Задатак базена нити се огледа у томе да преузме први објеката у мулти мапи, односно објекат са највећим приоритетом, а најмањим коефицијентом. На основу преузетог објекта се прозива одговарајући клијент и доставља му се вредност догађаја (Слика 3.5). По завршетку прозивања клијента, преузети објекат се брише из мулти мапе, при чему остају само објекти догађаја, односно клијенти, који још нису прозвани. У случају да се сви клијенти прозову, односно сви објекти мулти мапе избаце, базен нити прелази у стање мировања, и чека да пристигне нови догађај.



Слика 3.5 Архитектура прозивања клијента други део

4. Програмско решење

У овом поглављу дат је опис имплементације распоређивача догађаја као додатак имплементације вишенитног сервиса написаног у Ц++ програмском језику. Додатно је дат опис платформе која је била коришћена у инфозабавном систему возила.

4.1 Циљна платформа – Qualcomm SA8155P

Qualcomm SA8155 представља моћну и свестрану хардверску платформу дизајнирану специјално за уређаје са Андроид оперативним системом (Слика 4.1). Развијена од стране компаније Qualcomm, водећег произвођача полупроводника и телекомуникационе опреме, ова платформа пружа ефикасно решење са испуњавањем високих перформанси за широк асортиман апликација.

У самом срцу плоче се налази процесор Qualcomm Snapdragon 855. Изграђен на 7nm технологије производње, Snapdragon 855 нуди изузетну перформансу и енергетску ефикасност, чинећи га идеалним за потребе Андроид уређаја. Процесор поседује осмојезгарну конфигурацију, која комбинује три кластера са Kryo 485 процесором.

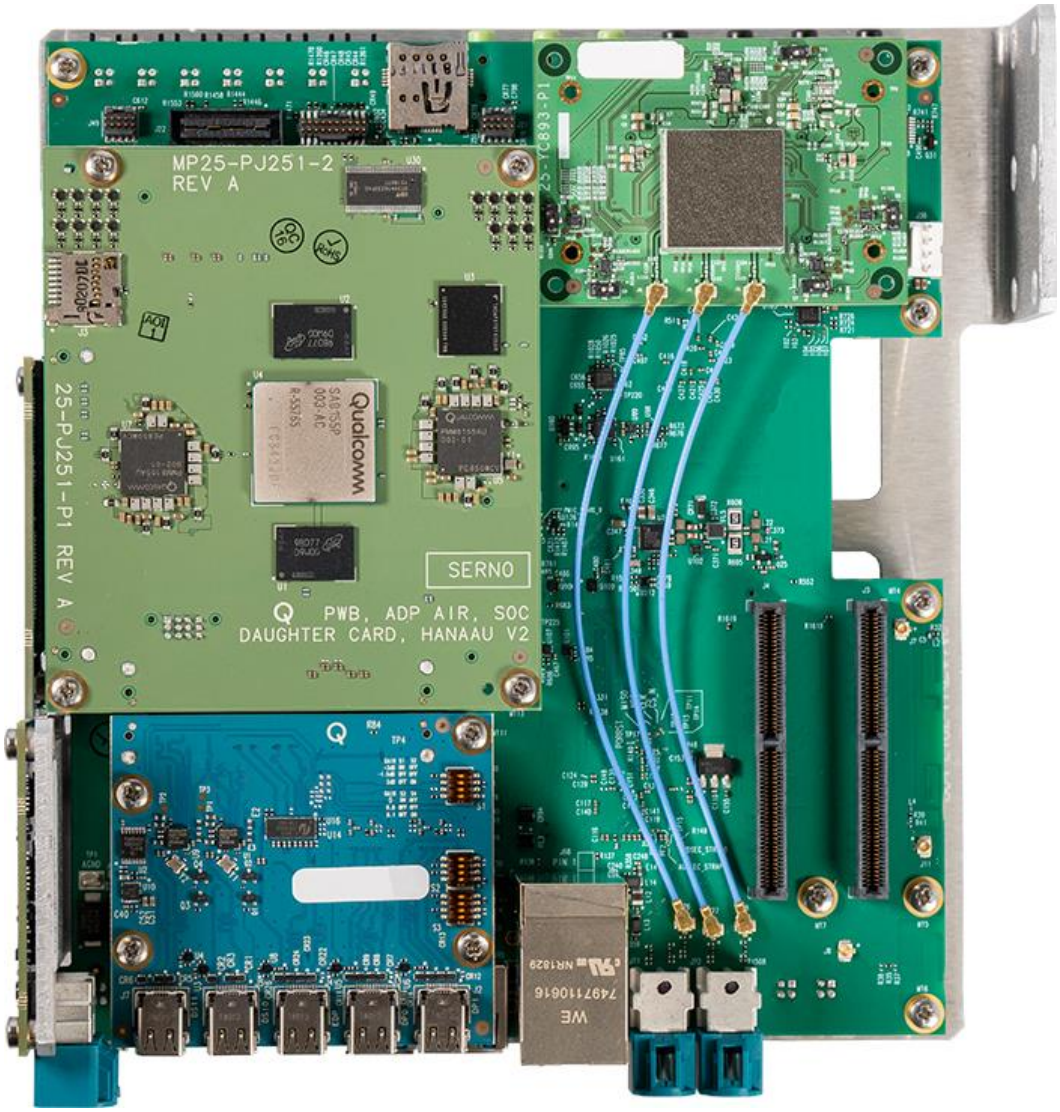
Поред моћног процесора, Qualcomm SA8155 плоча, такође интегрише Adreno 640 графичку процесну јединицу. Adreno је дизајниран да пружи задивљујућу графику и глатко искуство у игрању игрица, што га чини погодним за висококвалитетне игрице и мултимедијалне апликације. Подржава напредне технике исцртавање, укључујући HDR исцртавање и реално време праћења зрака, омогућавајући програмерима да креирају визуелно импресивна искуства.

SA8155 плоча нуди бројне опције повезивања ради подршке различитим комуникационим стандардима. Интегрира мулти-модем за 4G LTE, пружајући брзу и поуздану мобилну конекцију за пренос података и гласовну комуникацију. Плоча такође

подржава Wi-Fi 6 за бежично повезивање високих брзина и Блутут 5.0 за упаривање уређаја за трансфер података.

Да би се олакшао развој и прилагођавање, плоча пружа свеобухватан сет алата за развој и софтверске ресурсе. Програмери могу искористити SNPE да оптимизују и убрзају вештачку интелигенцију и машинско учење на плочи. Додатно, Qualcomm-ов Hexagon DSP омогућава ефикасну обраду аудио, сензорских и рачунарских визуелних података.

Укупно, Qualcomm SA8155 плоча нуди снажну и функционалну богату платформу за уређаје базиране на Андроиду. Њен моћни процесор, напредну графичко процесну јединицу и широк спектар опција повезивања чине је погодном за различите уређаје, укључујући паметне телефоне, таблете, паметне телевизоре и инфозабавне уређаје.



Слика 4.1 Плоча Qualcomm SA8155P

4.2 Имплементација распоређивача

У фокусу рада је имплементација ефикасног распоређивача догађаја у Андроид сервису написаном у Ц++ програмском језику. Специфично, истражени су проблематика и изазови који се јављају при обради догађаја у систему коју функционише у реалном времену, као што је инфозабавни систем у аутомобилу. Један од кључних проблема са којима се суочавамо, јесте обезбеђивање да се догађаји изврше унутар неког одређеног временског оквира.

Како би се решио овај изазов, развио сам решење које убрзава и оптимизује обраду догађаја. Централна тачка приступа је уградња распоређивача догађаја са базеном нити унутар Андроид сервиса. Овај механизам омогућава паралелно извршавање догађаја, дистрибуирајући их на расположиве нити система, један од улазних параметара система је и број хардверских нити приликом креирања базена нити. Ова паралелизација доводи до значајног побољшања перформанси система као и до смањења времена обраде догађаја.

Као допринос, развијен је приступ који користи ЕНМ за ефикасно распоређивање догађаја. Истраживање је показало да је ЕНМ од суштинске важности за брзу и прецизну обраду догађаја. Кроз детаљне експерименте и анализу перформанси, утврдио сам да ово решење знатно побољшава време одзива система и смањује време обраде догађаја у поређењу са традиционалним приступом.

Овај рад има широку примену у доменима где је потребно ефикасно управљање догађајима посебно у системима базираним на Андроиду.

Имплементација Андроид сервиса се састоји од централне компоненте за комуникацију са клијентом, помоћних класа са складиштење клијената, базена нити са улогом прозивања клијената и ЕНМ за креирање и распоређивање објеката догађаја.

4.2.1 Централна компонента сервиса

У самом центру сервиса налази се имплементација која има улогу директне комуникације са клијентом. Улога му је у добављању позива од стране клијената као и пружање спреге за комуникацију са клијентом. Ова компонента садржи објекат Јава помоћног сервиса, за комуникацију са клијентима у случају превеликог броја слушаоци, уместо директног прозивања клијената, биће послат Интент са одређеним именом, а клијенти су дужни да се претплате на њега и прихвате вредности које су дистрибуиране њиме. Такође садржи објекте за складиштење клијената за сваки догађај о коме ће бити више речи касније. На крају, најбитније, налазе се све методе декларисане у AIDL за комуникацију са клијентом, тачније, редефинисане методе из AIDL спреге.

4.2.2 Компонента за складиштење

Улога компоненте за складиштење почиње од момента када се прозове главна централна компонента сервиса од стране клијента. С обзиром да сервис поседује објекте ове компоненте за сваки догађај, над сваким објектом се адекватно позивају методе. Ова компонента представља шаблон класу, независно од типа догађаја или типа вредности који се пропагира, био то примитиван тип или неки сложен, попут Parcalable, имплементација је увек иста. Компонента садржи методе за додавање клијената и за брисање клијената, које представљају комуникацију са централном сервисном компонентом.

SomeipEventSubscriber()	Конструктор класе
int8_t registerEventObject(pid_t callingPID, uid_t callingUID, const std::shared_ptr<ListenerType>& listener, int64_t (*subscribeFunction)(int32_t, void (*callback)(const ArgumentType &)), void (*callbackFunction)(const ArgumentType &), void (*onBinderDied)(void *), int32_t instanceID);	Метода за претплату и проверу дозвола клијента
int8_t unregisterEventObject(pid_t callingPID, uid_t callingUID, const std::shared_ptr<ListenerType>& listener, int8_t (*unsubscribeFunction)(int32_t, uint32_t &), int32_t instanceID);	Метода за прекид претплате и проверу дозвола клијената
int8_t storeListener(const std::shared_ptr<ListenerType>& listener, pid_t callingPID);	Приватна метода за додавање клијента и прављење објекта терминираног повезивача
int8_t removeListener(const std::shared_ptr<ListenerType>& listener);	Приватна метода за брисање клијента и брисање објекта терминираног повезивача
void subscribeIfNeeded(int64_t (*subscribeFunction)(int32_t, void (*listener)(const ArgumentType &)), void (*callbackFunction)(const ArgumentType &),	Метода за претплату на екстерни сервис и добављање догађаја

void (*onBinderDied)(void *), int32_t instanceID);	
void unsubscribeIfNeeded(int8_t (*unsubscribeFunction)(int32_t, uint32_t &), int32_t instanceID);	Метода за прекид претплате на екстерни сервис за добављање догађаја
void handleDeadBinder(const void* deadBinder);	Метода која се прозива ако се установи да је клијент завршио извршавање
std::vector<NicenessListener<ListenerType> > m_vectorListeners;	Вектор објеката спреге клијента са толерантност вредношћу
AIBinder_DeathRecipient* m_deathRecipient	Објекат примања терминираног повезивача
std::mutex m_mtx	Мутекс
std::condition_variable m_cv	Условна променљива
bool m_ready = true	Променљива за проверу стања условне променљиве
uint32_t m_subscriptionCode	Повратна вредност претплате на екстерни сервис, за потребе прекидање претплате

Улога методе за додавање се огледа у провери да ли клијент који комуницира са сервисом поседује одговарајуће дозволе, којима обезбеђује примање податка. Када клијент прође проверу, провера се да ли је послат адекватан тип клијената, да не би дошло до нарушавања рада система. Након свих извршених провера, позивају се две приватне методе, прва има улогу претплате на неки екстерни сервис који ће доставити догађај, наравно ако има потребе за тим, такође ће направити објекат терминираног повезивача из безбедносних мера, у случају да се животни век клијента завршио, тај клијент ће аутоматски бити избачен из складишта клијента посредством тог објекта. Друга метода представља саму срж додавања односно складиштења клијента. За почетак, преузима вредност толерантности од објекта клијента, улога те вредности је у додели приоритета односно распоређивању клијената о чему ће бити више речи касније, која се спаја са објектом клијента у посебан објекат који ће бити сачуван. За случај да гранични број клијената није достигнут, нови објекат клијента, који садржи и толерантност и сам објекат клијентске спреге, ће бити сачуван, додатно се користи објекат терминираног повезивача, даје му се повратна метода

која ће се прозвати и доставља му се повезивач клијента, који се користи за проверу завршетка рада клијента. Након целе путање извршавања, клијент ће бити обавештен о статусу додавања, као и о могућој грешци која се изазвала.

Имплементација брисања у оквиру компоненте за брисање се огледа у методи која ће, као и метода за чување клијената, прво да провери да ли клијент који позива методу има захтевана дозволе, као и да ли је одговарајући клијент послат, након чега се зову две интерне методе компоненте за складиштење клијената. Прва метода је само избацивање клијената из складишта, прво се претражују сви клијенти, помоћу објекта клијентског повезивача, када наиђемо на инстанцу клијента, избацује се из контејнера, и такође се избацује објекат повезивача који је коришћен за потребе објекта терминираног повезивача. Након успешног избацивања клијента, позива се потенцијално прекидање претплате на екстерни сервис који је достављао догађаје, за случај је претходно избачен последњи сачувани клијент тог типа догађаја, чиме умањујемо оптерећење самог сервиса.

4.2.3 EHM

Имплементација распоређивача догађаја представља важан корак у постизању ефикасног и добро организованог система за рад у реалном времену, уз комбинацију различитих типова распоређивања, чини једну прилагодљиву компоненту система у складу са специфичним потребама.

Улога компоненте за распоређивање почиње од момента пристизања новог догађаја. За почетак пролази кроз све складиштене клијенте и за сваког клијента, креира се објекат помоћне класе, која се може посматрати као објекат догађаја. Помоћна класа је шаблонског типа, функционише исто за сваки тип догађаја и сваки тип вредности која се пропагира, била она сложеног или простог типа. Садржи поља као што су, време доласка догађаја, број необрађених објеката догађаја, бројач направљених објеката догађаја, и статичко поље које представља фреквенцију обраде догађаја, као аргументе који ће утицати на логику распоређивања. Затим објекат спреге клијента, која ће се користити за прослеђивање вредност догађаја, такође, садржи и конкретну вредност која се пропагира. За крај, поседује и две помоћне методе, односно два омотача функција, једну за прозивање клијентске методе из AIDL спреге, и другу који служи као спрега са компонентом за складиштење. У случају да се клијентска функција извршава дуже од очекиваног и договореног времена, о чему ће бити више речи касније, прозива се метода којом ће се клијент одстранити из складишта претплаћених клијената.

SomeipEvent(const std::shared_ptr<ListenerType>& listener,	Конструктор класе

ArgumentTypeAfterConversion arg, void (*listenerFun)(std::shared_ptr<ListenerType> , ArgumentTypeAfterConversion), void (*removeListenerFun)(std::shared_ptr<ListenerType>))	
static void calculateFrequency();	Метода за рачунање учесталости којом пристиже догађај
std::shared_ptr<ListenerType> m_Listener;	Дељени показивач на објекат спреге клијента
ArgumentTypeAfterConversion m_argument;	Вредност која се шаље догађајем, након конверзије у одговарајући тип који подржава AIDL спрега
Void (*ptrListenerFunction) (std::shared_ptr<ListenerType>, ArgumentTypeAfterConversion);	Показивач на функцију, која ће се прозвати када пристигне догађај
void (*ptrRemoveListenerFunction) (std::shared_ptr<ListenerType>);	Показивач на функцију за брисање објекта спреге клијента
std::chrono::high_resolution_clock::time_point m_arrivalTime;	Поље за временски моменат у коме пристигне догађај, за потребе рачунања коефицијента приоритета
static uint8_t m_remainingEventObjects;	Поље које показује колико клијената није обрађено у претходном догађају приликом пристизања текућег, за потребе рачунања коефицијента приоритета
static uint8_t m_eventCounter;	Поље бројач пристиглих догађаја, користи се за рачунање фреквенције пристизања догађаја
static uint8_t m_eventFrequency;	Поље фреквенције пристизања догађаја, за потребе рачунања коефицијента приоритета догађаја
static std::mutex m_mtx;	мутекс

Након креирања објекта догађаја за сваког клијента, прелази се на рачунање коефицијента који ће служити за распоређивање. Објекат који је управо направљен, служиће као вредност мулти-мапе, док ће кључ бити израчунат коефицијент приоритета. Разлог из ког је искоришћења мулти-мапа се огледа у два фактора, први је тај да догађаји буду сортирани од мањег ка већем коефицијенту, односно од приоритетнијег ка мање приоритетном догађају. Други разлог коришћена мулти-мапе је њена способност да постоје различите вредности мулти-мапе, које поседују исти кључ. Другим речима, могућност додавања објеката догађаја, иако већ постоји објекат догађаја са истим коефицијентом, што представља малу вероватноћу у пракси, али теоријски постоји та могућност, и не треба је избећи.

Коефицијент приоритета представља тип вредности са покретним зарезом. На њега утичу четири вредности. Са обзиром да радимо са системом за рад у реалном времену, један од основних захтева и циљева, који желимо да испунимо, се огледа у обради оног што је управо стигло за неко адекватно време. Из тог разлога први и фактор представља време доласка догађаја, који се задаје конструктору објекта догађаја. Време доласка је израчунато коришћењем часовника који мере време са високом прецизношћу. Рачуна се апсолутно време доласка догађаја, тренутно време догађаја се одузме од референтног времена почетка рада сервиса, чиме добијамо моменат у коме је пристигао догађај. као референтна тачка у времену се узима почетак рада сервиса.

Иако радимо на систему за рад у реалном времену, не желимо да се чисто секвенцијално прослеђују догађаји. Као вредност којом желимо да извршимо сепарацију између клијената коришћена је толерантност која проистиче из линукса, али је заступљена и у Андроид оперативном систему. Та вредност представља конкретни приоритет процеса који се извршава на уређају, који користи распоређивач језгра уређаја, из тог разлога представља адекватну вредност коју сам тражио за сепарацију процеса. Приликом складиштења клијента, узима се вредност толерантности од клијента, помоћу идентификационог броја процеса клијента. Та вредност се креће од минус двадесет па до деветнаест, нула представља неутрално стање процеса, а што је вредност мања, то је процес већег приоритета, иста логика је примењена и у имплементацији распоређивача.

Трећи фактор који утиче на коефицијент приоритета представља број необрађених објеката догађаја, као статичко поље објекта догађаја. У моменту када се пристигне догађај, и када се креирају објекти догађаја, конструктором се увећава бројач необрађених објеката догађаја, и сви објекти шаблонске класе истог типа ће видети исту вредност. У моменту када се догађај заврши и клијент буде успешно обавештен, вредност се умањује. Потреба за

оваквим фактором, се огледа у томе, да желимо да на време приметимо који тип догађаја се не обрађује довољно брзо, односно примећујемо пораст догађаја тог типа који пристиже од оног који се обрађује, из тог разлога, приоритет тог догађаја се повећава. Овим фактором се директно утиче на смањење загушења система, као и обезбеђивање да клијенти буду увек обавештени о свим вредностима без изузетака.

Последњи фактор представља учесталост пристизања догађаја. За разлику од трећег фактора, који представља вредност, која се увећава и смањује, када догађај пристигне односно када се обради, овај фактор представља долазак догађаја. Суштина је у томе, што ће се ова вредност у блоковским временским размацима, обновити и представити тренутну учесталост доласка. Сваким доласком догађаја, врши се провера, којом утврђујемо да ли је истекао временски оквир рачунања учесталости. У случају да је истекао, преузимамо бројаче догађаја и рачунамо учесталост за сваки тип догађаја. Временски оквир се помера, а бројачи догађаја се враћају у почетно стање. Након што се израчуна учесталост, добијена вредност се множи коефицијентом, добијеним од толерантности, што представља нормализацију толерантности. Највећи приоритет, односно вредност минус двадесет ће бити број један, средња вредност, односно нула ће бити презентовано као једна половина, док најмањи приоритет, односно деветнаест се представља као једна четрдесетина. Разлог произилази из потребе да догађај који пристиже учесталије, буде и обрађен са већим приоритетом од осталих догађаја. Чиме је испуњена потреба система за рад у реалном времену да догађаји буду обрађени у захтеваном времену са утицајем захтева корисника у виду коефицијента којим се множи учесталост.

Након што се сакупе сви фактори, на основу тестирања и детаљних експеримената и темељне анализе перформанси, дошао смо до одговарајућих коефицијената, којима се множи сваки фактор. Сваки фактор је помножен са адекватном вредношћу уз задржавање важности сваког приоритета, нисам желео да фактор који мање утиче, буде већег значаја од битнијег коефицијента. Сабирањем помножених фактора долазимо до коначног фактора догађаја. Коначни фактор ће се уз објекат догађаја, који садржи све потребне информације за прозивање клијената, додати у мулти-мапу и проследити базену нити за даљу обраду.

4.2.4 Базен нити

Након што су објекти догађаја креирани и адекватно додати у мулти-мапу, преостаје да се преузму и обраде до краја, што је улога базена нити, односно извршне компоненте. Као компонента сервиса, која мора бити стално у приправности, покреће се и поставља у почетно стање од стране сервиса, на почетку рада истог. Као спрегу са централном

компонентом сервиса, користи две методе, једну која ће покренути рад и једну која ће зауставити рад базена нити.

Прва нит на располагању ће представљати руководиоца базена нити, са улогом контролера, који ће се старати да комплетна компонента ради несметано и имаће контролу над дужином извршавања клијената. Остале нити ће бити резервисане за прозивање клијената и могу се посматрати као радне нити. Када се преузме број нити на хардверском уређају, заузима се меморија за вектор нити, коришћен за приступање одређеној нити која тренутно ради. Заузима се меморија за вектор временских момената, који ће представљати моменат у коме је започета акција прослеђивања вредности клијенту, за потребе контролисања извршавања клијената. Трећи вектор који се користи, представља вектор стања нити, да би контролер имао информацију о режиму рада сваке радне нити.

void start();	Метода за покретање базена нити
void stop();	Метода за прекидање базена нити
std::multimap<float, SomeipEvents> m_someipEventSubscribers;	Мулти мапа објеката догађаја за прозивање, кључ је коефицијент приоритета а вредност је објекат догађаја
std::map<int, SomeipEvents> m_someipEventSubscribersForThreads;	Мапа догађаја, кључ представља редни број нити која обрађује догађај, вредност је објеката догађаја
std::mutex m_mtxEventSubscribersMap;	Мутекс за приступ мулти мапи
std::condition_variable m_cvEventSubscribersMap;	Условна променљива за мулти мапу, користи је нит контролера
void threadLoop(uint8_t threadPosition);	Метода коју извршавају радне нити
void threadHandler(uint8_t numberOfThreads);	Метода коју извршава контролна нит
void createWorkingThreads(uint8_t numberOfThreads);	Метода који ја креира и започиње радне нити
void detachThread(uint8_t threadID);	Метода којом се ослобађа нит
void createNewThread();	Метода којом се креира нова радна нит и пропратне вредност

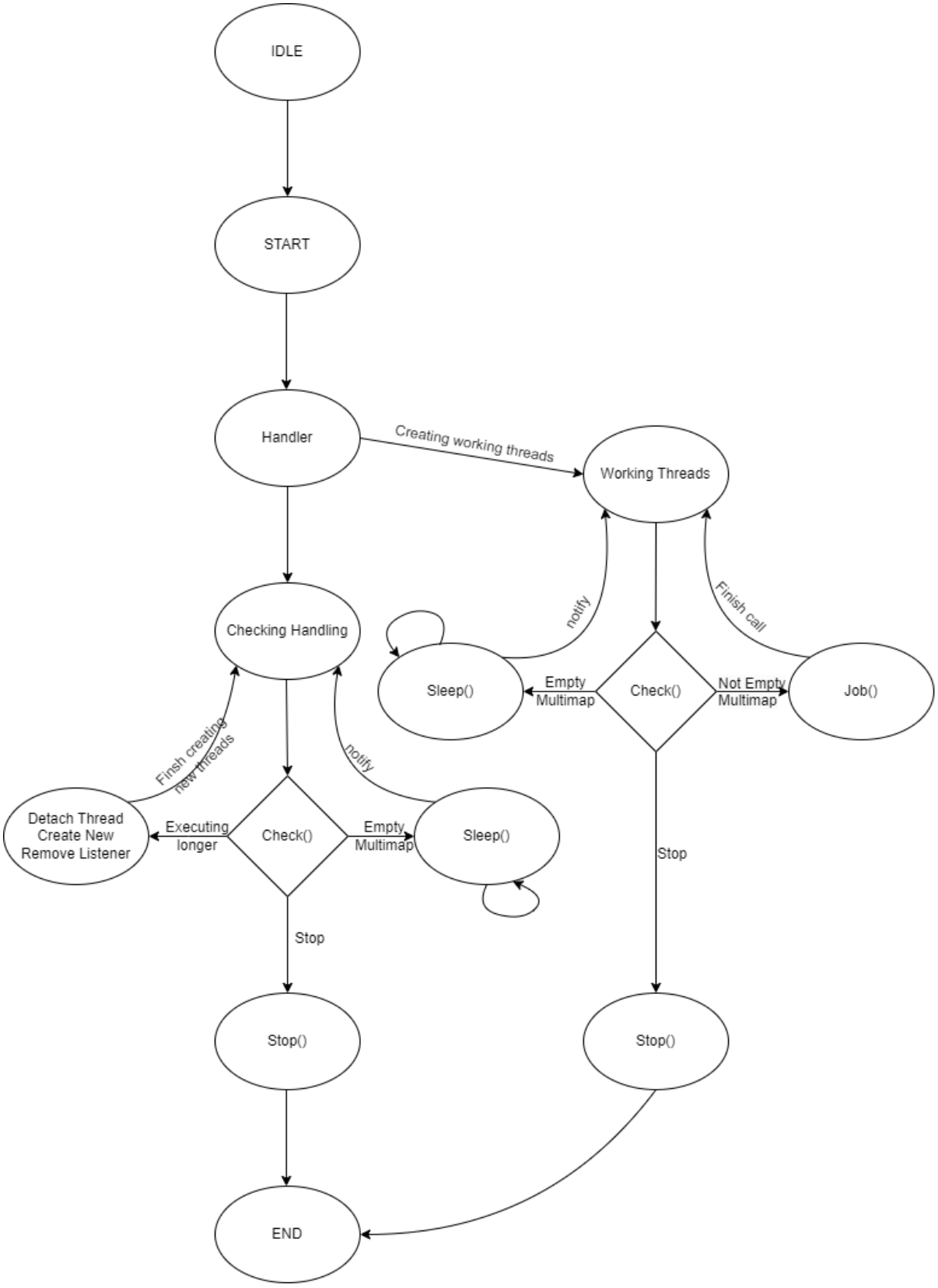
void stopWorkingThreads();	Метода којом се заустављају радне нити
void unregisterProblematicListeners(uint8_t threadID);	Метода који брише објекат спреге клијента, ако се извршавао дуже од очекиваног времена
bool m_shouldWorkingThreadTerminate = false;	Поље којим се контролише прекидање извршавања радних нити
bool m_shouldHandlerTerminate = true;	Поље којим се контролише извршавање контролне нити
int8_t m_numberOfWorkingThreads = 0;	Поље које представља тренутни број активних нити
std::chrono::microseconds m_maxJobExecutionTime = std::chrono::microseconds(20000);	Поље које представља временско ограничење извршавања клијента
std::mutex m_mtxVectorThreads;	Мутекс за приступање, додавање и брисање над вектором нити
std::mutex m_mtxVectorThreadState;	Мутекс за приступ и промену вектора стања нити
std::condition_variable m_cvThreadLoop;	Условна променљива за потребе извршних нити
std::vector<std::thread> m_vectorThreads;	Вектор нити
std::vector<std::chrono::high_resolution_clock::time_point> m_vectorTimePoints;	Вектор временских момената, време рада извршавања клијената по нити
std::vector<int8_t> m_vectorThreadState;	Вектор стања нити, може бити активно, ослобођено и стање мировања

Прва покренута нит је нит контролера, која ће покренути појединачно сваку радну нит да извршава свој посао. Проверава стање активних нити, у случају да је установљено да све радне нити спавају, проверава да ли је преостао неки посао да се изврши, односно проверава испуњеност мулти-мапе, у случају да је празна, контролер прелази у стање чекања заједно са радним нитима. Из стања чекања, биће пробуђен у случају да пристигне нови догађај и мулти-мапа крене да се попуњава. Након свог буђења, дужан је да пробуди све радне нити

које тренутно чекају. У случају да неке радне нити извршавају свој посао, или да мулти-мапа није празна, он је дужан да проверава времена извршавања клијентских функција. Проласком кроз вектора нити, и провером да ли су тренутно у радном режиму, преузима временски моменат у којем је започет посао и врши проверу са својим локалним временом. Одузимањем почетка прозивања клијентске функције од локалног времена, и провером да ли је премашено очекивано време извршавања, дужан је да регулише рад система. За почетак, мења стање те нити у вектору, са радног режима, на режим ослобођене дужности, у моменту када се заврши посао, радна нит ће бити свесна свог стања, о томе ћемо касније. Након тога, покреће процес раскидања везе са том нити, у случају да упадне у блокирајуће стање, језгро уређаја ће се постарати о прекидању рада те нити, чиме наш систем остаје стабилан. Пошто смо изгубили један ресурс, односно једну радну нит, контролер ће се постарати о покретању нове нити, као и додавањем исте у вектор нити, и попуњавање свих вектора са иницијалном вредности за ту нит. Последњи корак, представља комуникација са компонентом за складиштење клијената. Помоћу помоћне методе из објекта догађаја, о коме је било речи у делу за распоређивање догађаја, прозива се избацивање проблематичног клијента, чиме је спречено даље блокирање и нарушавање рада система од стране проблематичног клијента.

Имплементација радних нити, започиње провером стања мулти-мапе. Мулти-мапа представља поље базена нити, попуњава се од стране компоненте распоређивача догађаја, међусобно искључивост је адекватно испоштована. У случаја да је мулти-мапа празна, радна нит ће прећи у стање чекања и чекаће да на буђење од стране контролне нити. Када се пробуди преузима се први објекат из мулти-мапе, односно објекат највећег приоритета, и додаје се у помоћну мапу за прозивање, кључ представља редни број нити под којом је нит покренута, тако да не долази до мешања послова између радних нити. Након што се дода објекат у помоћну мапу, тај исти објекат се избацује из мулти-мапе, умањује се статичко поље, коефицијент необрађених догађаја из објекта догађаја, и ослобађа се мулти-мапа. Након тога се мења стање у вектору стања нити, на радни режим и поставља се тренутни временски моменат у вектору са временским моментима за одговарајућу нит. Након што су сва одговарајућа поља вектора постављена, прозива се функција из објекта клијента, односно прослеђује се вредност од догађаја ка клијенту посредством објекта спреге који је прослеђен приликом претплате на сервис. Након завршетка функције клијента, проверава се стање нити, ако нит региструје да је стање постављено на раскинуту вредност од стране контроле, нит се прекида и завршава своје извршавање, с обзиром да је нова нит већ започета за рад. У случају да стање није раскинуто, враћа се назад и провера стање мулти-мапе, и понавља се цела процедура.

У случају да сервис завршава свој рад, позива се метода за прекидање базена нити. Сви вектори и сва заузета меморија се ослобађа, чекају се нити које су у радном режиму на завршетак и на спајање са главном нити, након чега се спаја и контролна нит и тиме се завршава животни циклус базена нити (Слика 4.2).



Слика 4.2 Аутомат стања базена нити

5. Евалуација

Потврда имплементације распоређивача догађаја у оквиру Андроид сервиса извршена је испитивањем на уређају Qualcomm SA8155P. Као екстерни систем који је производио догађаје коришћена је плоча за потребе аутомобилске индустрије, Jetson AGX Xavier.

Тестирање се може поделити у две целине. Прва целина представља тестирање изолованих делова система, коришћењем Гугл тестова (енг. GUnit). Испитане су и верификоване методе, поновљеним поступцима са очекиваним вредностима и очекиваним понашањем система у стотинама итерацијама. Потенцијални исходи извршених Гугл тестова су прошли (резултат се поклапа са очекиваним понашањем система) или пали (резултат се не поклапа са очекиваним понашањем система). Укупно је урађено и успешно извршено сто двадесет осам тестова за сервис у Ц++ програмском језику са екстензијом распоређивача догађаја и базеном нити. Тестови су подељени на следећи начин:

- осамнаест тестова - су потврдили да је комуникација између сервиса и клијента успешна и да је клијент успео да добије вредност од стране сервиса
- тридесет пет тестова - су служили као безбедносна провера за највећи могући број клијената и тестирање понашање система
- двадесет тестова - су верификовали да само одговарајући клијенти могу да успоставе комуникацију са сервисом и да се претплате на догађај
- пет тестова - су потврдили да различите комбинације претплата не нарушавају систем
- двадесет пет тестова – су верификовали рад базена нити и брисање клијента у случају да се клијентска метода дуже извршавала
- тридесет пет тестова – су потврдили очекивано понашање система у случају прекидања претплате клијента

Другу део тестирања су чинили функционални тестови. Направљене су Андроид апликације које су представљале клијенте за наш сервис. Коришћена су два типа догађаја који су се пропагирали, догађаје мале и велике учесталости, са простим и сложеним типом вредност која се прослеђује. Док се за број клијената који се регистровао на сервис, користио један, пет , десет, тридесет и педесет. Такође је вршена комбинација три различита типа догађаја који су пристизали у исто време.

Кроз Гугл тестове и функционалне тестове успели смо да потврдимо рад нашег система у различитим случајевима, различитој оптерећености.

Кроз први пример мерења, направљено је окружење у коме је измерен временски одзив система и потребна времена за случај рада са једним догађајем. Направљени су клијенти у виду Андроид апликација, који су се претплатили на сервис и очекивали догађај. Мерење је извршено кроз четири циклуса са различитим бројем клијената. Први случај представља једног клијента претплаћеног на сервис, други представља пет клијената, трећи десет и последњи тестирани случај представља тридесет клијената који су истовремено претплаћени на исти догађај. У Табела 5.1 се могу видети забележена времена, као што су просечно време, медијана време, минимално, максимално време у јединици милисекунде, као и стандардна девијација у процентима. Свако време представља време које је потребно да се достави информација до тестираног броја клијената. У Табела 5.2 су представљени коришћени ресурси за случај једног догађаја. Табела приказује искоришћење централне процесорске јединице у процентима, заузеће виртуелне меморије у гигабајтима, као и потребна физичка и дељења меморија у мегабајтима, са приказом процената потребне меморије.

Један активан догађај					
Број клијената	Просечно време[ms]	Медијана време[ms]	Максимално време[ms]	Минимално време[ms]	Стандардна девијација[%]
1	1.665539409	1.61675	8.1835	0.843	22.915
5	3.143701039	2.8145	13.940	1.387	39.978
10	4.495278568	4.0125	17.1715	1.0015	39.444
30	6.160101071	5.7705	22.5075	0.9665	31.541

Табела 5.1 Временски одзив система за један догађај

Један активан догађај					
Број клијената	Централна процесорска јединица[%]	Виртуелна меморија[Gb]	Физичка меморија[Mb]	Дељена меморија[Mb]	Меморија[%]
1	23	11	37	32	0.3
5	30.6	11	37	32	0.3
10	47	11	37	32	0.3
30	87	11	37	32	0.3

Табела 5.2 Потрошња ресурса система за један догађај

Кроз други пример мерења, поновљен је први корак, са додатком још једног догађаја на који су клијенти били претплаћени. Поновљен је исти број клијената са два различита догађаја. Број клијената је варирао од један, пет, десет и тридесет. У Табела 5.3 се виде резултати мерења као што су просечно, медијана време, највеће и најмање време извршавања одређеног броја клијената, заједно са стандардном девијацијом. У Табела 5.4 се могу видети резултати коришћених ресурса за овај вид мерења. Такође су приказани, као и у првом случају, процесорско заузеће, виртуелна, физичка и дељена меморија, као и меморија у процентима.

Два активна догађаја					
Број клијената	Просечно време[ms]	Медијана време[ms]	Максимално време[ms]	Минимално време[ms]	Стандардна девијација[%]
1	1.835472146	1.667	6.855	0.6785	36.497
5	3.233430505	2.7905	13.6995	1.0525	46.795
10	3.223571697	2.41475	14.0195	0.7935	66.381
30	8.673828533	6.9015	57.117	0.4535	83.619

Табела 5.3 Временски одзив система за два догађаја

Два активна догађаја					
Број клијената	Централна процесорска јединица[%]	Виртуелна меморија[Gb]	Физичка меморија[Mb]	Дељена меморија[Mb]	Меморија[%]
1	27.6	11	37	32	0.3
5	46.3	11	37	32	0.3

10	73	11	37	32	0.3
30	149	11	38	32	0.3

Табела 5.4 Потрошња ресурса система за два догађаја

Последњи пример мерења, представља комбинацију првог и другог случаја мерења, односно укупно три могућа догађаја на који су се клијенти претплатили. Поновљен је исти број клијената са два различита догађаја. Број клијената је варирао од један, пет, десет и тридесет. У Табела 5.5 се виде резултати мерења као што су просечно, медијана време, највеће и најмање време извршавања одређеног броја клијената и стандардна девијација у процентима. У табели2 се могу видети резултати коришћених ресурса за овај вид мерења.

Три активна догађаја					
Број клијената	Просечно време[ms]	Медијана време[ms]	Максимално време[ms]	Минимално време[ms]	Стандардна девијација[%]
1	1.755180074	1.623	10.304	0.459	38.186
5	3.54756	3.1465	14.309	1.1935	47.073
10	5.84861	5.22525	24.615	20.065	44.669
30	2312489	2483.500	3936.895	35.7935	44.908

Табела 5.5 Временски одзив система за три догађаја

Три активна догађаја					
Број клијената	Централна процесорска јединица[%]	Виртуелна меморија[Gb]	Физичка меморија[Mb]	Дељена меморија[Mb]	Меморија[%]
1	32.6	11	37	32	0.3
5	65.3	12	38	32	0.3
10	81	11	37	32	0.3
30	151	11	39	32	0.3

Табела 5.6 Потрошња ресурса система за три догађаја

Кроз евалуацију и анализу, примећене су значајне варијације у временима одзива и потрошњи ресурса при различитим оптерећењима система. Имплементацијом сервиса, омогућено је, да у случају стандардног извршавања не успева да испуни постављене захтеве, могуће је ефикасно реаговати помоћу Интената. У том случају, сервис не би директно комуницирао са клијентима посредством AIDL спреге, већ би поседовао само

једног клијента у јава програмском језику. Клијент би се понашао као вид проширења сервиса, са улогом прављења одговарајућег Интента, са вредношћу коју добије од сервиса. Након што се креира, Интент бива дистрибуиран у етар, а клијенти су у могућности добавити га помоћу имена Интента, без потребе познавања сервиса, као и без претплате на сервис. Ово проширење омогућава боље искоришћење ресурса и обезбеђује оптимално функционисање система, чак и у захтевним ситуацијама. Просечно време, измерено у стотинама итерација, које је потребно да се дистрибуира Интент и да клијент одреагује на њега је око десет милисекунди.

Поред тога што се ово решење показало као веома корисну у окружењима, где је критично одржавати одређени редослед извршавања, пружити сигурност и гаранцију пристизања података, уводи кашњење у целокупан систем. У случају да корисник не жели да има сигурност достављања догађаја, нити могућност доделе приоритета ни распоређивања догађаја, систем ће се знатно брже показати у временима одзива. Овакав начин функционисања показао се као мање поуздан у критичним ситуацијама, с обзиром да сервис нема никакву контролу ни потврду и примљеним подацима са стране клијента. У Табела 5.7 се може приметити како изгледа просечно, медијана, максимално и минимално време одзива у јединици милисекунде, као и стандардна девијација у процентима. Као тестирано окружење, узет је један активан догађај и комбинација једног, пет, десет и тридесет претплаћених клијената.

Један активан догађај					
Број клијената	Просечно време[ms]	Медијана време[ms]	Максимално време[ms]	Минимално време[ms]	Стандардна девијација[%]
1	0.489	0.498	1.388	0.187	23.945
5	1.464	1.463	3.830	0.712	27.630
10	2.308	1.971	6.631	0.699	35.663
30	4.759	4.831	11.603	1.820	23.378

Табела 5.7 Временски одзив система за један догађај без ЕНМ и базена нити

6. Закључак

У овом раду сам се фокусирао на имплементацију распоређивача догађаја и базен нити као додатак Андроид сервису написаном у Ц++ програмском језику. Циљ је био да побољшам перформансе сервиса и ефикасно управљам вишенитним извршавањем послова. На овај начин инфозабавном систему, који је базиран на Андроиду, је омогућено прикупљање информација, распоређивање истих и обраду у стриктно дефинисаним временским роковима.

Кроз овај рад, успео сам да постигнем значајно побољшање у перформансама сервиса, овај приступ је омогућио ефикасно распоређивање догађаја и планирање њиховог извршавања, да би се задовољиле потребе система за рад у реалном времену.

Поред тога што сам успео да ефикасно управљам и распоређујем догађаје, имплементација базена нити је омогућила паралелно извршавање послова, што је резултирало бољом искоришћеношћу ресурса и смањењем времена извршавања.

Ово решење је могуће поставити и користити на сваком уређају који је заснован на Андроид оперативном систему. Решење је евалуирано на две верзије Андроид оперативног система, односно Андроид 11 и Андроид 12.

Решење је испитано на Qualcomm SA8155 циљној платформи са Андроид оперативним системом. Тестирање је извршено Гугл тестовима и потврђене су следеће функционалности сервиса :

- Успешна комуникација између Андроид клијента и Андроид сервиса, посредством AIDL спреге и претплаћивањем на сервис
- Складиштење свих клијената на сервис страни
- Успешно распоређивање догађаја и прилагођавање у реалном времену
- Успешно прозивање клијената приликом по приоритету

Даље истраживање и оптимизација овог приступа може довести до додатних побољшања у перформансама и ефикасности Андроид сервиса. Неки даљи напредак и побољшање се огледа у комбинацији са и без ЕНМ и базена нити. Прецизније, комбинације споријег, али безбеднијег система са распоредом приоритета и вишенитног извршавања и бржег система који не пружа сигурност пропагирања података догађаја, без распореда приоритета. Још један могући напредак се огледа у комбинацији ЕНМ и Интената. Тачније, могућност система да препозна у ком моменту извршавање и одзив система постаје критичан, те да се аутоматски пребаци на комуникацију са Интентом и помоћним Јава сервисом, такође и да препозна моменат у ком може да се врати да првобитни начин комуникације. У будућности се надамо да ће ова имплементација бити коришћена за испуњавање очекиваног рада система у реалном времену, у погледу перформанси, ефикасности и строгих временских ограничења.

7. Литература

[1] D. Wu, D. He, S. Chen and J. Xue, "Exposing Android Event-Based Races by Selective Branch Instrumentation," 2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE), Coimbra, Portugal, 2020, pp. 265-276, doi: 10.1109/ISSRE5003.2020.00033.

[2] Zhenhua Huang and Qingyun Dai, "The complex event processing mechanism for RFID real-time data on Android platform," 2014 IEEE International Conference on Consumer Electronics - China, Shenzhen, 2014, pp. 1-2, doi: 10.1109/ICCE-China.2014.7029881.

[3] V. Janarthanan, P. Gohari and A. Saffar, "Formalizing real-time scheduling using priority-based supervisory control of discrete-event systems," in IEEE Transactions on Automatic Control, vol. 51, no. 6, pp. 1053-1058, June 2006, doi: 10.1109/TAC.2006.876806.

[4] S. W. Keckler, A. Chang, W. S. L. S. Chatterjee and W. J. Dally, "Concurrent event handling through multithreading," in IEEE Transactions on Computers, vol. 48, no. 9, pp. 903-916, Sept. 1999, doi: 10.1109/12.795220.

[5] L. Bilac, D. Stanisic, D. Kenjic and M. Antic, "One Solution of an Android In-Vehicle Infotainment Service for Communication with Advanced Driver Assistance System," 2022 45th Jubilee International Convention on Information, Communication and Electronic Technology (MIPRO), Opatija, Croatia, 2022, pp. 1420-1425, doi: 10.23919/MIPRO55190.2022.9803790.

[6] Liu, B., Zhang, C., Gong, G., Zeng, Y., Ruan, H., & Zhuge, J. (2020). {FANS}: Fuzzing Android Native System Services via Automated Interface Analysis. In 29th USENIX Security Symposium (USENIX Security 20) (pp. 307-323).

[7] Андроид архитектура сајт, https://www.androidauthority.com/aosp-explained-1093505/, учитано 26.06.2022.

[8] Pap dr Ištvan, Lukić dr Nemanja, „Projektovanje i arhitekture softverskih sistema – Sistemi zasnovani na Androidu“

[9] Q. Gan and H. Wu, "The Research of Android Broadcast Intercept Technology Based on Priority," 2012 Fourth International Conference on Multimedia Information Networking and Security, Nanjing, China, 2012, pp. 556-559, doi: 10.1109/MINES.2012.233.

[10] S. Nomura, Y. Nakamura, H. Sakamoto, S. Hamanaka and S. Yamaguchi, "Improving choice of processes to terminate in Android OS," 2014 IEEE 3rd Global Conference on Consumer Electronics (GCCE), Tokyo, Japan, 2014, pp. 624-625, doi: 10.1109/GCCE.2014.7031148

[11] Андроид процеси сајт, https://developer.android.com/guide/components/processes-and-threads, учитано 26.06.2022.

[12] AIDL сајт, https://developer.android.com/guide/components/aidl, учитано 26.06.2022.

[13] *Treble* пројекат сајт, https://android-developers.googleblog.com/2017/05/here-comes-treble-modular-base-for.html, учитано 24.06.2022.

[14] Повезивач сајт, https://source.android.com/devices/architecture/hidl/binder-ipc, учитано 26.06.2022.

[15] Позадивана повезивача сајт, https://source.android.com/devices/architecture/aidl/aidl-backends, учитано 26.06.2022.

[16] Zhang, L., Yang, Z., He, Y., Zhang, Z., Qian, Z., Hong, G., ... & Yang, M. (2018, October). Invetter: Locating insecure input validations in android services. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (pp. 1165-1178).

[17] Андроид Интенти сајт, https://developer.android.com/reference/android/content/Intent, учитано 22.06.2022.

[18] S. A. Widhiono, M. Ary Murti and C. Setianingsih, "Electronic Loads Control and Management Using Priority Queue Algorithm on Android Based Smartphone," 2019 International Seminar on Research of Information Technology and Intelligent Systems (ISRITI), Yogyakarta, Indonesia, 2019, pp. 154-159, doi: 10.1109/ISRITI48646.2019.9034600.