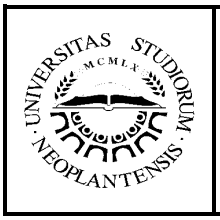


УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА

УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА
НОВИ САД
Департман за рачунарство и аутоматику
Одсек за рачунарску технику и рачунарске комуникације

ЗАВРШНИ (BACHELOR) РАД

Кандидат: **Јана Бодвански**
Број индекса: **РА 84/2020**

Тема рада: **Имплементација WAYLAND програмског клијента за управљање андроид виртуалном тастатуром**

Ментор рада: **Проф. др Илија Башичевић**

Нови Сад, јул, 2024

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**:		Завршни (Bachelor) рад
Аутор, **АУ**:		Јана Бодвански
Ментор, **МН**:		Проф. др Илија Башичевић
Наслов рада, **НР**:		Имплементација програмског клијента за управљање андроид виртуалном тастатуром
Језик публикације, **ЈП**:		Српски / латиница
Језик извода, **ЈИ**:		Српски
Земља публиковања, **ЗП**:		Република Србија
Уже географско подручје, **УГП**:		Војводина
Година, **ГО**:		2024
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)		7/29/3/26/1/0/0
Научна област, **НО**:		Електротехника и рачунарство
Научна дисциплина, **НД**:		Рачунарска техника
Предметна одредница/Кqучне речи, **ПО**:		„Wayland“ протокол, „Android“ оперативни систем, виртуелна тастатура
УДК		
Чува се, **ЧУ**:		У библиотеци Факултета техничких наука, Нови Сад
Важна напомена, **ВН**:		
Извод, **ИЗ**:		У оквиру задатка је имплементиран сервис за руковање улазима са тастатуре и њихово прослеђивање Андроид оперативном систему, који се налази у такозваном контејнеру. Сервис је задужен за обраду података као и за правилно мапирање вредности сваког тастера.
Датум прихватања теме, **ДП**:		
Датум одбране, **ДО**:		09.07.2024.
Чланови комисије, **КО**:	Председник:	Проф. др Мирослав Поповић
	Члан:	Проф. др Иван Каштелан — 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**:		Bachelor Thesis	
Author, **AU**:		Jana Bodvanski	
Mentor, **MN**:		Prof. dr Ilija Bašičević	
Title, **TI**:		Implementation of a Wayland program client for managing the Android virtual keyboard	
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**:		2024	
Publisher, **PB**:		Author's reprint	
Publication place, **PP**:		Novi Sad, Dositeja Obradovica sq. 6	
Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes)		7/29/3/26/1/0/0	
Scientific field, **SF**:		Electrical Engineering	
Scientific discipline, **SD**:		Computer Engineering, Engineering of Computer Based Systems	
Subject/Key words, **S/KW**:		Wayland protocol, Android operating system, virtual keyboard	
UC			
Holding data, **HD**:		The Library of Faculty of Technical Sciences, Novi Sad, Serbia	
Note, **N**:			
Abstract, **AB**:		Within the scope of this project, a service for handling keyboard inputs and forwarding them to the Android operating system, was implemented, which resides within a so-called container. The service is responsible for processing data and accurately mapping each key.	
Accepted by the Scientific Board on, **ASB**:			
Defended on, **DE**:		09.07.2024.	
Defended Board, **DB**:	President:	PhD Miroslav Popović	
	Member:	PhD Ivan Kaštelan	Menthor's sign
	Member, Mentor:	PhD Ilija Bašičević	

Захвалност

Захваљујем се академском ментору проф. др Илији Башичевићу на помоћи током израде овог рада. Посебно се захваљујем техничком ментору Саши Мудри, као и свим колегама из Мартини тима на корисним саветима и доступности. Такође, бих се захвалила породици и пријатељима на константној подршци током школовања, као и колегама са одсека на свакој помоћи и бодрењу у претходним годинама студирања.

САДРЖАЈ

Списак слика

Списак табела

Скраћенице

LXC – Linux Container (Линукс контејнер)

gRPC – Remote Procedure Calls (Даљински позиви процедура)

AOSP – Android Open Surce Code (Андроид пројекат отвореног кода)

WI-FI – Wireless Networking Technology (Бежична мрежна технологија)

USB – Universal Serial Bus (Универзална серијска магистрала)

IoT – Internet of Things (Интернет ствари)

IVI – In-Vehicle Infotainment (Информационо-забавни систем у возилу)

CPU – Central Processing Unit (Централна процесорска јединица)

RAM – Random-Access Memory (Меморија са случајним приступом)

GPU – Graphic Processing Unit (Графичка процесорска јединица)

APK – Android Application Package (Пакет Андроид апликације)

GNSS – Global Savigation Satellite System (Глобални навигациони сателитски систем)

АПИ – Application Programming Interface (Површина за програмирање апликација)

ID – Identification Number (Идентификациони број)

1. Увод

Убрзани развој аутомобилске индустрије прати и растућа потражња за системима који повећавају привлачност аутомобила на тржишту. У овакве системе спадају и infotainment системи који укључују велики број функција као што су: интеграција са паметним телефонима, навигација, гласовне команде итд. Они комбинују информационе и забавне функције како би возачима и путницима пружили боље корисничко искуство.

Martini пројекат се бави унапређењем таквог једног система у оквиру кога је могуће покретати Андроид апликације. Андроид оперативни систем се у овом случају

покреће коришћењем контејнерске технологије, тачније коришћењем LXC контејнера. Такође, користи се и Yocto Open Source пројекат који пружа алате и процесе за креирање прилагођених Линукс дистрибуција за уграђене системе.

Задатак овог рада јесте да се на оперативном систему домаћина (Линукс) имплементира Wayland програмски клијент за управљање Андроид виртуалном тастатуром и омогући његово даљинско коришћење у Андроид оперативном систему као замена за постојећу апликацију задужену за контролу употребе виртуалне тастатуре. Wayland је протокол графичког приказа за Линукс радну површину и библиотека која имплементира тај протокол.

Рад се састоји од седам поглавља. Прво поглавље представља увод у завршни рад, где је циљ да се представе основе које су битне за креирање иницијалне идеје о раду. Друго поглавље се односи на теоријске основе које чине темељ овог рада. У трећем поглављу је описан концепт решења које се у највећој мери односи на ток израде задатка. Четврто поглавље чини опис програмског решења, у ком се обједињују теоријске основе и идеје о имплементацији овог задатка. У петом поглављу су представљени резултати израде овог рада, док је у шестом поглављу, које је названо "Закључак", извршен преглед главних момената у раду, као и дискусија о њиховом значају за даља истраживања и практичне примене. У седмом поглављу је наведена коришћена литература.

2. Теоријске основе

2.1 Martini програмско решење

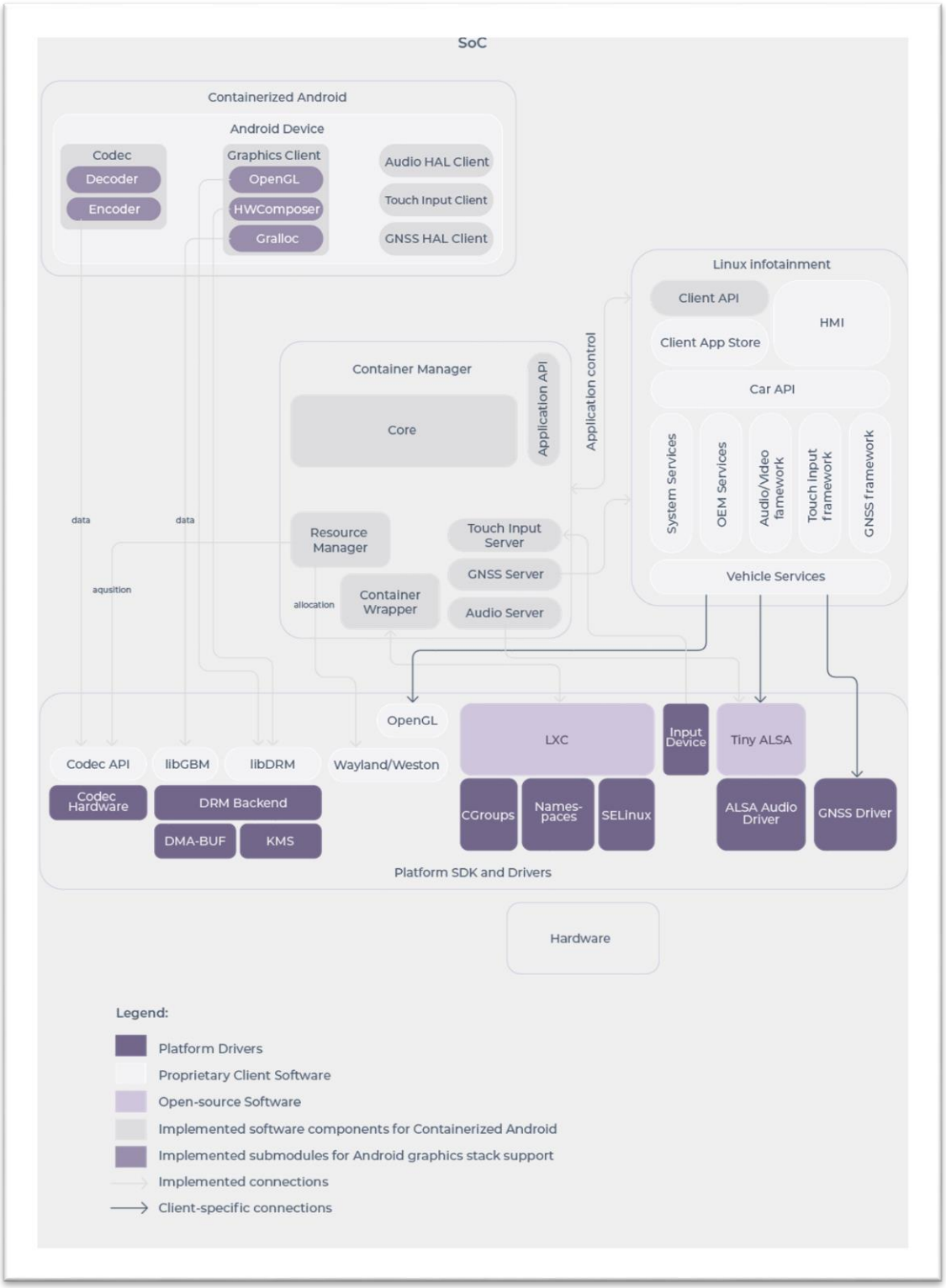
Будући да је овај задатак рађен у оквиру Martini програмског решења, најпре је неопходно описати неколико кључних момената које се односе на теоријски део овог програма.

Као што је већ напоменуто у уводном делу, Martini програмско решење представља мултимедијални систем за возило. Главни циљ јесте дизајнирање и развој системског решења које омогућава покретање Андроид и Линукс IVI апликација у истом системском окружењу. Андроид који користи контејнерску технологију омогућава покретање Андроид апликација у контролисаном окружењу унутар Линукс оперативног система. Наиме, могуће је динамички алоцирати (додељивати) системске ресурсе као што су CPU, RAM и GPU. Сходно томе, оба оперативна система могу извршавати захтевније апликације или оптимизовати ресурсе, чиме се постиже смањење трошкова који се тичу хардвера, без утицаја на перформансе. Кључне компоненте овог програмског решења и њихов кратак опис, дате су у следећој табели:

Линукс “infotainment”	Управљач контејнерима	Андроид уређај
- садржи подсистем главне јединице - специфициран од стране произвођаче оригиналне опреме (OEM)	- аутоматски конфигурише контекст извршавања контејнера - управљање ресурсима - модул аудио сервера - модул сервера за ГНСС	- прилагођени АОСП уређај - подешавање Андроид окружења - модул клијента за аудио - модул клијента за ГНСС

Табела 1 – Кључне компоненте Martini програмског решења

У наставку је приказана архитектура Martini система коју прати кратко објашњење о њеном радном току.



Слика 1 – Архитектура Martini програмског решења [1]

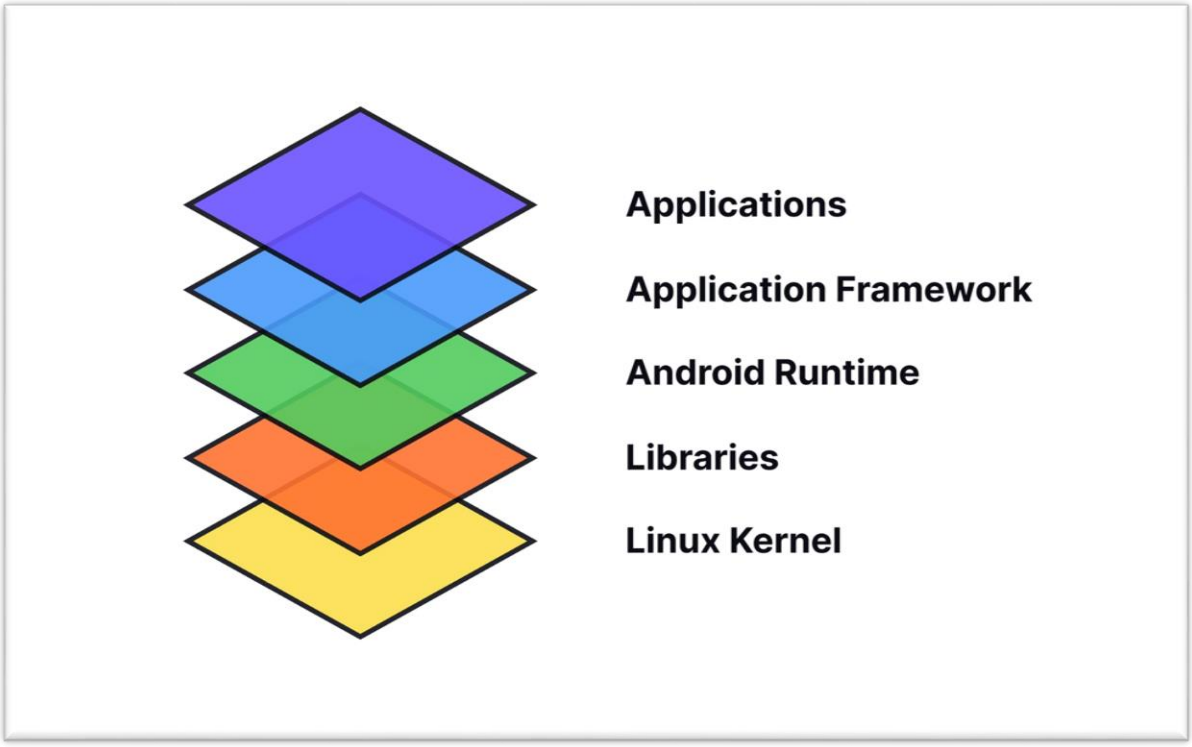
На почетку радног тока, контејнеризовани Андроид је присутан у позадини, а модули комуникације између оперативних система се аутоматски иницијализују. При покретању, у систему не постоји ниједна Андроид апликација, због чега корисник мора изабрати која Андроид апликација му је потребна. Захтев за инсталацију апликације се иницира коришћењем специјално имплементираног клијентског АПИ-ја, док се инсталација апликације на контејнеризовани Андроид систем врши преко управљача контејнера. Исто тако, када апликација треба да се покрене, захтев за покретање се иницира коришћењем имплементираног клијентског АПИ-ја. Апликација се затим покреће на Андроиду. Прозор активности се представља на Линукс оперативном систему, преко WAYLAND сервера за приказ, са подршком за догађаје који се односе на додиривање екрана.

2.2 Андроид оперативни систем

Андроид је оперативни систем који је развила компанија Гугл. Базиран је на Линукс кернелу и дизајниран је првенствено за уређаје са екраном осетљивим на додир, као што су паметни телефони и таблети. Такође, користи се и на разним наменским уређајима који имају специфичне функције и примене. Андроид је погодан за овакве уређаје због својих карактеристика које доприносе флексибилности и глобалној распрострањености овог оперативног система [2]. Неке од њих су:

- отвореност и прилагодљивост (Андроид је део пројекта отвореног кода)
- разноврсност апликација (велики број апликација које се могу лако преузети са Google Play продавнице)
- интуитивна спрега (eng. interface) (корисничка спрега која се лако може прилагодити)
- способност повезивања (подржава разне начине повезивања као што су WI-FI, Bluetooth , мобилни подаци (3G, 4G, 5G), USB итд.)
- интеграција са различитим Гугл сервисима

Андроид архитектура је такође значајна, јер због своје природе омогућава проширене функционалности широком спектру уређаја. Архитектура овог оперативног система је дизајнирана тако да се лако може прилагодити различитим типовима уређаја са различитим хардверским захтевима и наменама. Андроид је изграђен на Линукс кернелу, који управља основним системским услугама као што су управљање меморијом, управљање процесима, мрежна повезаност и управљање хардверским уређајима.



Слика 2 – Приказ Андроид архитектуре по слојевима [3]

Због употребе Линукс кернела, Андроид може управљати разним хардверским компонентама као што су сензори, камере и многи други уређаји. Сигурносне фунцкије које су интегрисане у Линукс кернел и Андроид Runtime слој знатно помажу у заштити података и система, што је за данашњицу посебно важно. Андроид Application Framework(апликациони радни оквир) и Runtime слојеви омогућавају ефикасно управљање системским ресурсима, као што су меморија и процесорска снага. Ова карактеристика Андроид архитектуре је кључна за наменске уређаје који морају да раде глатко и поуздано у реалном времену. Библиотеке које су подржане у Андроиду пружају функционалности као што су рендеровање графике, базе података, као и различите мултимедијалне библиотеке за репродукцију аудио и видео садржаја. Поред тога, Андроид подржава и интеграцију са облаком (Cloud) и IoT платформама.

2.2.1 AOSP

АОСП (Android Open Source Project) је пројекат са отвореним кодом који представља почетну тачку за развој Андроид оперативног система. Он обезбеђује изворни код, алате за развој и упутства за изградњу Андроида. Такође, често се користи за креирање прилагођене верзије Андроида за различите уређаје и потребе. Управо ову улогу је АОСП заузео у Martini пројекту.

2.3 Линукс контејнер

Линукс контејнерска технологија је веома битна у овом пројекту, због чега је потребно представити објашњење о концепту рада.

Ова технологија омогућава извршавање целог оперативног система или издвојених процеса на оперативном систему домаћина, слично виртуалним машинама. Међутим, за разлику од виртуалне машине, уместо да креирају цео виртуелни оперативни систем, контејнери не морају да пролазе кроз читав тај процес, већ креирају само појединачне компоненте које су им потребне за рад. Рад контејнера је много бржи, јер за разлику од традиционалне виртуализације, процес се у суштини извршава на домаћину, само са додатним слојем заштите око њега. У овом пројекту оперативни систем домаћина је Линукс, у оквиру кога се покреће контејнер симболично назван Martini. Једна од најбитнијих особина ове технологије јесте изолација апликација од оперативног система који је домаћин. Сваки контејнер поседује свој сопствени кориснички простор, библиотеке, систем за управљање датотекама и мрежне конфигурације. Контејнери омогућавају програмеру да „упакује“ апликацију са свим деловима који су јој потребни и да је испоручи у једној целини. Једна од имплементација ове технологије је LXC контејнер и управо она се користи у овом задатку.

2.4 Yocto Open Source Project

Yocto је Линуксов пројекат чији је циљ да произведе алате и процесе који омогућавају креирање Линукс дистрибуција на широком спектру хардверских архитектура. Овај пројекат је отвореног типа, што значи да постоји отворени код, слободан за коришћење. Компоненте и њихове функционалности су један од кључних разлога због којих се програмери одлучују за његово коришћење.

Неке од њих су:

- Bitbake (главни алат за изградњу система који управља процесом превођења и интеграције софтвера)
- Poky (референтна платформа која обухвата Bitbake, шаблоне, метаподатке и документацију за креирање прилагођених Линукс слика)
- Devtool (пружа механизаме за брзо и једноставно додавање, измењивање и превођење софтверских компоненти, као и за надгледање њихових измена)

Неопходно је споменути и Yocto Build који обухвата превођење свих компоненти дистрибуције, укључујући кернел, библиотеке, апликације и конфигурације, како би се креирала финална слика за уграђени систем.

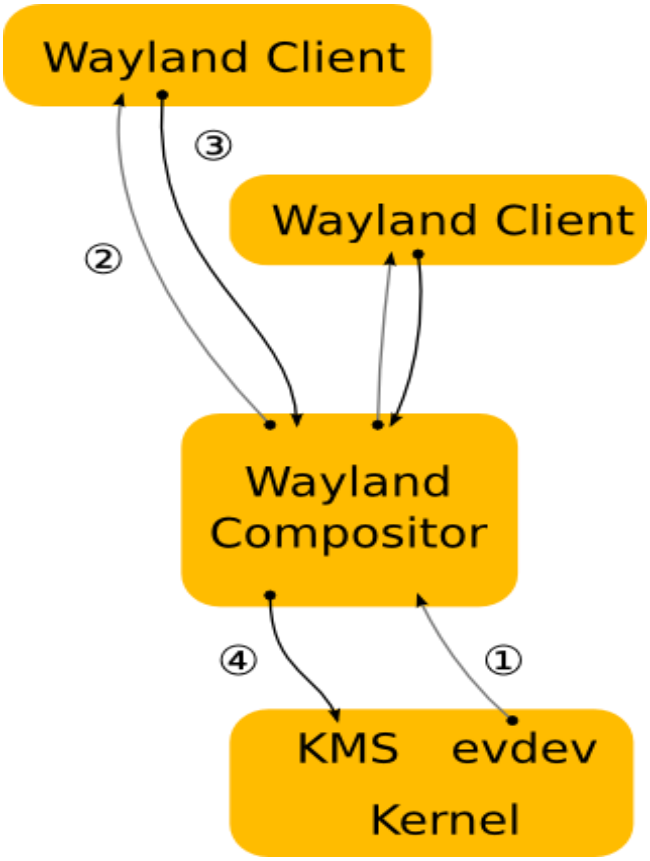
2.5 Wayland протокол

Wayland је протокол којег апликације могу користити за комуникацију са сервером екрана како би постале видљиве и добиле унос од корисника.Сервер екрана који користи Wayland протокол се назива Wayland композитор, јер обавља задатак композиције управљача прозора. Wayland се односи и на системску архитектуру, а не само на однос сервер-клијент између композитора и апликација. Не постоји један заједнички Wayland сервер, већ свако графичко окружење доноси једну од многих имплементација композитора.

Централна компонента архитектуре је libwayland - библиотека за међупроцесну комуникацију. Ова библиотека не имплементира протокол, већ само обрађује поруке. Део Wayland пројекта је и референтна имплементација Wayland композитора, што значи да је то програм који примењује Wayland протокол за управљање приказом на екрану и манипулацију прозорима.

2.5.1 Wayland архитектура

1. Кернел прима догађај и шаље га композитору.
2. Композитор врши преглед могућих случајева и одређује који прозор треба да прими догађај.
3. Рендеровање се дешава у клијенту и клијент једноставно шаље захтев композитору, указјући на регион који је ажуриран.
4. Композитор прикупља податке о променама од својих клијената, а затим поново генерише екран.



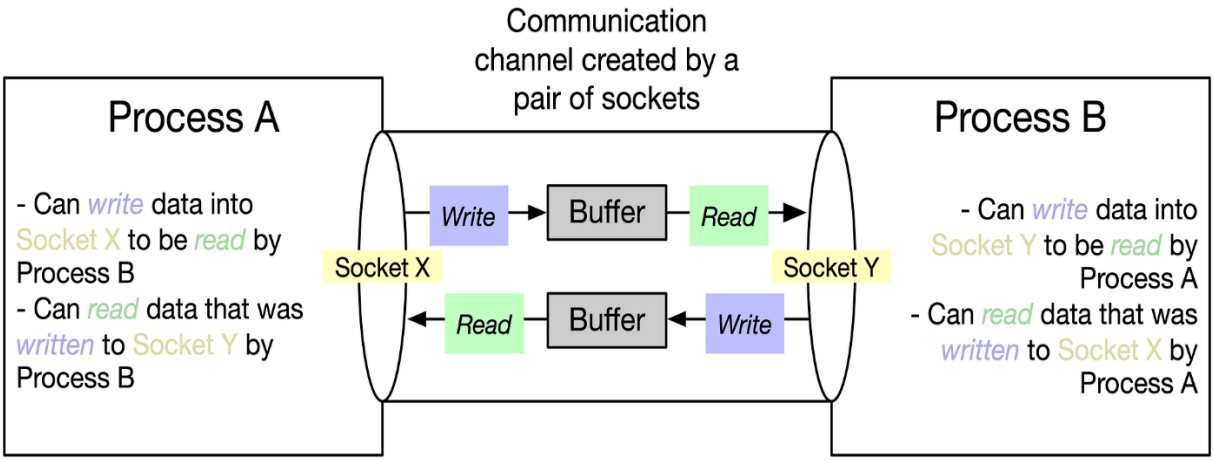
Слика 3 – Wayland архитектура [4]

2.5.2 Обрада уноса у Wayland протоколу

За обраду уноса у овом протоколу се користе такозвана „седишта“ (“seats”). Седиште пружа апстракцију над догађајима koji se, у Wayland-у, односе на унос података. Такорећи, Wayland седиште односи се на седиште на којем корисник „седи“ и користи рачунар, и повезује се са највише једним тастатуром и највише једним уређајем за показивање.

2.6 UDS – Unix Domain Socket

UDS (утичница за међупроцесну комуникацију) је крајња тачка за комуникацију података између процеса који се извршавају на истом оперативном систему који је домаћин. У изради овог задатка ова врста утичнице је коришћена за комуникацију између сервиса који управља обрадом уноса и Андроида који је покренут у контејнеру. Клијент је у овом случају сервис, а сервер је у Андроиду и он је задужен за слушање и примање података од стране клијента. UDS користи локалну датотеку на уређају у коју се смешта утичница, која је најпре креирана од стране сервера. У клијенту је потребно навести путању до те датотеке, како би и он могао да користи утичницу. Не захтева отварање мрежних портова, већ Линукс сам контролише ко може имати приступ датотеци за комуникацију. Овакан начин комуникације се обавља у оба смера, што је веома битна чињеница за овај рад.



Слика 4 – Приказ UDS комуникације између два процеса [5]

3. Концепт решења

Осмишљавање тока израде овог задатка започето је са успостављањем основних и неопходних алата и окружења. Претходно споменут Yocto Build и остали пропратни алати су преузети и смештени на рачунар на коме је рађен задатак, где је инсталиран Линукс оперативни систем. Када је Yocto преведен, уследило је преузимање, а затим и покретање Qemu емулатора који је у процесу израде кориштен уместо плоче. Qemu емулатор се покреће помоћу већ спремне скрипте у којој се налазе разни параметри потребни за прецизно дефинисање функционалности. Такође, потребно је и да се обави повезивање помоћу ssh конекције. Након успешног повезивања, Qemu терминал се користи за покретање контејнера у коме ће се затим покретати Андроид. Тада је први пут подигнут Martini програм, при чему се могло још боље упознати са системом због визуелне компоненте. Следећи неопходан корак је преузимање АОСП-а и његово превођење (eng. build). Када је то успешно завршено, добијене слике (system.img и vendor.img) се копирају у Yocto пројекат. Сваки следећи пут, када се буду правиле измене у АОСП-у, биће потребно копирати слике како би све измене биле видљиве при покретању.

Након успешног упознавања са базичним компонентама и функционалностима, омогућено је креирање идеје о томе како би решење могло изгледати, као и шта је све потребно за израду.

Први корак у изради представља сама имплементација Wayland клијента на Линукс оперативном систему рачунара. Било је потребно направити сервис тако да прима улаз са физичке тастатуре, обради га и пошаље серверу. Обрада улаза подразумева мапирање вредности притиснутог тастера на одговарајући карактер.

Касније ће бити непходно да се имплементирани клијент пребаци на Qemu, јер се у овом случају ту налази оперативни систем који је домаћин.

Овакав редослед је обавезан због тога што се у реалним околностима подразумева да се програм налази на плочи, а не на емулатору. Проверу почетне верзије имплементације је било могуће одмах обавити, због већ уграђеног Wayland композитора на оперативном систему рачунара.

Како би се обрађени улази проследили тамо где треба, било је непоходно пронаћи моменат у АОСП коду где се позива виртуелна тастатура у оквиру Андроида.

Идеја је била да се уклони тај моменат позивања, односно да се захтев преусмери на сервис који се налази на Линуксу. Тада би главну улогу имао направљен сервис за обраду улаза, који би комуницирао са Андроидом како би се руковање тим делом програма преусмерило у потпуности на спољњи сервис. Следеће питање које је решавано се односило на комуникацију имећу имплементираног сервиса (Wayland клијента) и Андроида који се налази у контејнеру. Након испробавања неколико техника као што су цев (eng. pipe) и дељени директоријуми, закључено је да би најбоље решење било коришћење утичница. Протокол који је зове Unix Domain Socket је изабран за комуникацију између два процеса. Овај протокол је такође искориштен и за преусмеравање захтева за активацију имплентираног сервиса. Пошто се овај протокол заснива на клијент-сервер односу, њих је првенствено било потребно имплементирати и проверити њихов рад успостављањем комункације. Пролазак кроз овај процес омогућио је даље интегрисање UDS клијента у сервис за тастатуру и UDS сервера у Андроид код. Идеја је била да се на овај начин преносе унешени и обрађени карактери из сервиса у Андроид где напослетку и треба да буду видљиви.

Један од проблема при комуникацији био је видљивост и пренос направљене утичнице. Када сервер направи утичницу, она мора да буде видљива и од стране клијента како би и он могао да је користи. Будући да се клијент налази на Линуксу (на Qemu емулатору), а сервер у контејнеру, било је неопходно урадити прикључивање (eng. mount) директоријума у коме ће се направити утичница. Детаљно објашњење ове врсте дељења директоријума ће бити објашњено у следећем поглављу.

Као што је већ споменуто, непходно је и пребацити сервис за тастатуру на Qemu. Да би било омогућено превођење у Yocto пројекту, најпре је креиран CMake како би се добиле све неопходне датотеке. Затим се у Martini програмско решење копирају CMake и датотека која садржи имплементацију сервиса за тастатуру и додатно се прави рецепт за покретање овог сервиса. Овим је омогућено његово покретање на циљаној платформи.

4. Програмско решење

Следи детаљан опис реализованих функционалности које су укратко представљене у претходном поглављу.

4.1 Покретање и превођење

За почетак ће бити наведене команде са којима се покрећу коришћени и имплементирани програми.

Првенствено се мора подесити окружење у коме се Андроид билдује командом source build/envsetup.sh. Затим је потребно идентификовати циљну платформу за превођење коришћењем lunch команде.

```
bjana@rtrkw624-lin:~/android_manifest$ lunch 32

============================================
PLATFORM_VERSION_CODENAME=REL
PLATFORM_VERSION=10
TARGET_PRODUCT=mARTini_qemu_x86_64
TARGET_BUILD_VARIANT=eng
TARGET_BUILD_TYPE=release
TARGET_ARCH=x86_64
TARGET_ARCH_VARIANT=x86_64
TARGET_2ND_ARCH=x86
TARGET_2ND_ARCH_VARIANT=x86_64
HOST_ARCH=x86_64
HOST_2ND_ARCH=x86
HOST_OS=linux
HOST_OS_EXTRA=Linux-6.5.0-35-generic-x86_64-Ubuntu-22.04.4-LTS
HOST_CROSS_OS=windows
HOST_CROSS_ARCH=x86
HOST_CROSS_2ND_ARCH=x86_64
HOST_BUILD_TYPE=release
BUILD_ID=QP1A.191105.004
OUT_DIR=out
============================================
```

Слика 5 – Бирање циљне платформе под редним бројем 32 помоћу команде lunch

Након тога Андроид је могуће превести и то командом m -j4.

Слике које су настале као резлтат превођења Андроида сада се копирају у одговарајући директоријум који се налази на путањи: martini_manifest/yocto-build/qemu/yocto-x86/meta-martini/meta-martini-core/recipes-misc/android/files.

Након овога Yocto је спреман за превођење покретањем команде bitbake core-image-weston, са претходно подешеним алатима: source poky/oe-init-build-env

Следећи корак је покретање Qemu помоћу скрипте: ./run_qemu.sh, у оквиру чега се покреће и скрипта за стварање мрежног окружења: ./tap_ifup.sh. Такође, постоји и скрипта која се покреће након коришћења Qemu, како би се обрисало мрежно окружење које се више не користи: sudo ./tap_ifdown.sh <if_name>.

При покретању скрипте се отвара прозор који је приказан на слици испод.

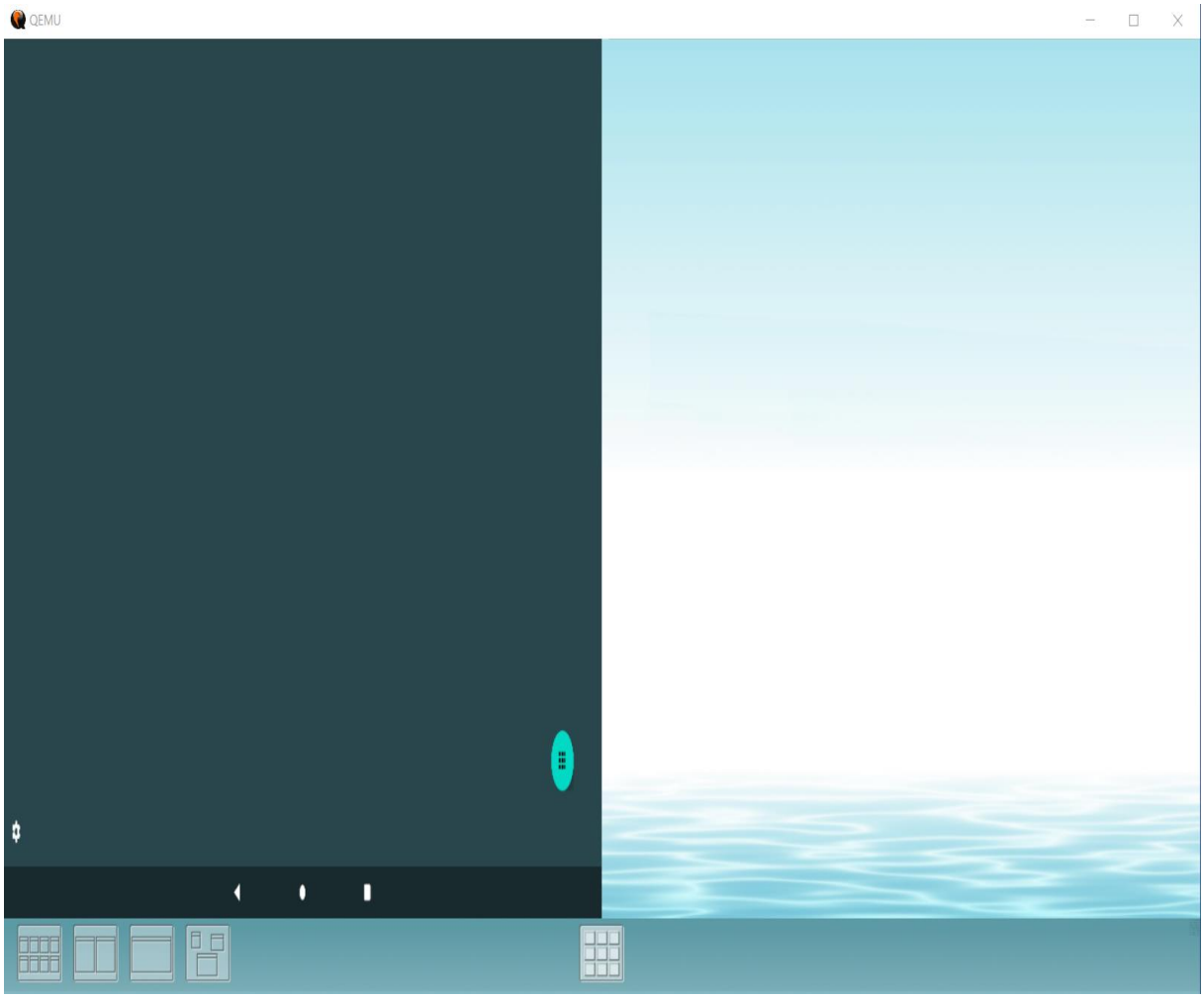
Слика 6 – Приказ почетног прозора у Martini програму

Како би оперисање из Qemu било омогућено, неопходно је успоставити конекцију са Qemu машином помоћу команде: ssh root@192.168.7.2. Наведена адреса се очитава у претходном кораку.

```
lbilac@rtrkw953-lin:/WORKAREAS/lbilac/ognjenreviewQEMU/QEMU-yocto/images_for_qemu$ ./run_qemu.sh
TAP: tap0

sudo qemu-system-x86_64 -enable-kvm -device virtio-net-pci,netdev=net0,mac=52:54:00:12:34:02 -netdev tap,id=net0,ifname=t
ap0,script=no,downscript=no -object rng-random,filename=/dev/urandom,id=rng0 -device virtio-rng-pci,rng=rng0 -drive file=
core-image-weston-qemux86-64.ext4,if=virtio,format=raw -usb -device usb-tablet -cpu host -machine q35 -smp 8 -m 10G -seri
al mon:vc -serial null -device virtio-vga -display sdl,show-cursor=on -kernel bzImage-qemux86-64.bin -append 'root=/dev/v
da rw ip=192.168.7.2::192.168.7.1:255.255.255.0::eth0:off:8.8.8.8 oprofile.timer=1 tsc=reliable no_timer_check rcupdate.r
cu_expedited=1 enforcing=0'
```

Слика 7 - Приказ терминала при покретању скрипте за Qemu

Остало је још да се покрене контејнер са командом „lxc-start -n martini“ као и да се уради повезивање на покренути контејнер „lxc-attach martini“ како би се унутар њега могле извршавати команде. Тада се може приказати Андроид прозор: setprop martini.show0 true.



Слика 8 – Андроид прозор у оквиру Martini система

4.2 Имплементација Wayland клијента

Wayland протокол ради по моделу клијент-сервер. Клијенти су обично апликације које комуницирају са корисником, док је сервер Wayland композитор који узима излазе од клијената, саставља их у приказ и показује их на неком екрану. У овом задатку имплементираће се Wayland клијент који у себи садржи функционалности везане за обраду улаза са тастатуре, као и прослеђивањем података на Андроид. За имплементацију кориштен је Wayland API.

Иницијални корак је подразумевао повезивање са композитором помоћу функције wl_display_connect(), чиме се шаље порука композитору. На ове поруке ће се одговарати, углавном асинхроно. Клијент управља својим одговорима путем руковалаца (eng. handler) или слушалица (eng. listener) које су регистроване у оквиру система. Када се користи wl_display_roundtrip() може се очекивати синхрони одговор, који ће блокирати процес док не пристигне одговор од сервера. Наравно, овај позив се мора користити опрезно, јер може значајно успорити систем или у најгорем случају може довести до застоја. Сервер контролише бројне објекте којима се приступа преко регистра. Да би се приступило објекту који дозвољава клијенту да региструје слушаоце за глобалне објекте на серверу, користи се функција wl_display_get_registry(). Ово је кључни корак у комуникацији јер омогућава клијенту да добије информације о доступним глобалним објектима, који уствари представљају различите сервисе и ресурсе које сервер нуди. Два слушаоца се додају овом позиву помоћу wl_registry_add_listener().

```c
int main(void)
{
    struct wl_display *display = wl_display_connect(NULL);
    if (display == NULL) {
        fprintf(stderr, "Failed to connect to Wayland display\n");
        return 1;
    }

    struct wl_registry *registry = wl_display_get_registry(display);
    if (!registry) {
        fprintf(stderr, "Failed to get Wayland registry\n");
        wl_display_disconnect(display);
        return 1;
    }

    wl_registry_add_listener(registry, &registry_listener, NULL);

    wl_display_dispatch(display);
    // wait for the "initial" set of globals to appear
    wl_display_roundtrip(display);
```

Слика 9 – Функције за успостављање комуникације између клијента и сервера

Слушаоци су функције, једна за нове заступничке(proxy) објекте и друга за њихово уклањање. Оне су обе упаковане у структуру типа wl_registry_listener која заправо садржи обе функције. Клијент не може много да уради док не добије заступника за важне објекте попут композитора. Сходно томе, има смисла направити блокирајући позив за повратак у регистар како би се добили објекти регистра.

Регистарски објекат има име, спрегу и целобројни идентификатор (ID). Идентификатор се користи у даљим позивима, док спрега омогућава програму да идентификује који је објекат регистра у питању. Заступник за регистарски објекат се добија везивањем идентификатора за одговарајући тип података (као што је например wl_compositor_interface).

```
void registry_global_handler
(
    void *data,
    struct wl_registry *registry,
    uint32_t name,
    const char *interface,
    uint32_t version
) {
    if (strcmp(interface, "wl_compositor") == 0) {
        compositor = wl_registry_bind(registry, name,
            &wl_compositor_interface, 3);
    } else if (strcmp(interface, "wl_shm") == 0) {
        shm = wl_registry_bind(registry, name,
            &wl_shm_interface, 1);
    } else if (strcmp(interface, "wl_seat") == 0) {
        seat = wl_registry_bind(registry, name,
            &wl_seat_interface, 1);
    } else if (strcmp(interface, "ivi_application") == 0) {
        ivi_application = (struct ivi_application *)(wl_registry_bind(registry, name, &ivi_application_interface, 1));
    } else if (strcmp(interface, "ivi_wm") == 0) {
        iviWm = (struct ivi_wm *)(wl_registry_bind(registry, name, &ivi_wm_interface, 1));
    }
}

void registry_global_remove_handler
(
    void *data,
    struct wl_registry *registry,
    uint32_t name
) {}

const struct wl_registry_listener registry_listener = {
    .global = registry_global_handler,
    .global_remove = registry_global_remove_handler
};
```

Слика 10 – Функције за регистрацију и руковање објектима који се налазе на серверу

Функција wl_registry_bind() служи управо за везивање за специфичне спреге са којима клијент треба да комуницира. Додатно је одрађено везивање за још неке спреге, као што су:

- wl_shm (Wayland Shared Memory/Дељена меморија)
- wl_seat (Wayland Seat/Седиште)
- ivi_application (In-Vehicle Infotainment Application/Апликација за информационо-забавне системе у возилу)
- ivi_wm (In-Vehicle Infotainment Window Manager/Менаџер прозора за информационо-забавне системе у возилу)

На слици 10 се такође може уочити да функционалност уклањања објеката није имплементирана јер се потребни објекти региструју само једном на почетку и не очекује се да ће бити уклоњени или промењени током рада програма.

4.2.1 Дељена меморија (wl_shm)

Проширење протокола wl_shm омогућава клијентима да креирају дељене меморијске бафере који се могу користити за цртање графичког садржаја. Бафери су обично мапирани у адресни простор клијента и њима се може директно манипулисати. На следећој слици приказано је креирање и мапирање бафера у меморији.

```
int width = 200;
int height = 200;
int stride = width * 4;
int size = stride * height;  // bytes */

// open an anonymous file and write some zero bytes to it
int fd = syscall(SYS_memfd_create, "buffer", 0);
ftruncate(fd, size);

// map it to the memory
unsigned char *data = mmap(NULL, size, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);

// turn it into a shared memory pool
struct wl_shm_pool *pool = wl_shm_create_pool(shm, fd, size);

// allocate the buffer in that pool
struct wl_buffer *buffer = wl_shm_pool_create_buffer(pool,
    0, width, height, stride, WL_SHM_FORMAT_ARGB8888);
```

Слика 11 – Креирање и мапирање дељеног бафера

Најпре се дефинишу димензије бафера у пикселима, а величина у бајтовима. Затим се креира анонимна датотека која се може користити као бафер, а која постоји само у меморији (није видљива у систему датотека). Помоћу ftruncate() функције се одређује колико ће меморије бити резервисано за сам бафер. Након тога се датотека мапира у адресни простор процеса, што омогућава да бафер буде директно доступан у меморији. Користе се и меморијски резервоари из којих ће се креирати бафер.

У овом случају дељена меморија се искористила да би се креирао бафер за приказ графичког садржаја. Клијент мапира бафер, црта у њега и затим обавештава композитора да је бафер спреман за приказивање. Композитор онда приступа том баферу и користи га за приказивање на екрану. На овај начин се омогућава ефикасна размена података између клијента и композитора без потребе за копирањем података.

4.2.2 Wayland седишта (wl_seat)

Wayland седиште се у овој имплементацији користи за добијање објекта који представља тастатуру повезану са одређеним wl_seat објектом. Функција wl_seat_get_keyboard() омогућава приступ улазним догађајима са тастатуре (попут притиска и отпуштања тастера) преко Wayland протокола.

```
//KEYBOARD
keyboard = wl_seat_get_keyboard(seat);
```

Слика 12 – Позив функције за добијање објекта тастатуре

4.2.3 Информационо-забавни систем у возилу (IVI)

Да би се у Wayland-у приказао било какав графички садржај неопходно је користити surface објекат. Surface(површина) је апстракција која представља правоугаону област у којој се може цртати графички садржај. Ову област можемо замислити као платно на којем апликације могу „рендеровати“ своје корисничке спреге или било које визуелне елементе. Композитор управља свим површинама и задужен је за њихово комбиновање и приказивање на екрану. [6]

У овој имплементацији, wl_surface је кориштен као основни објекат за прављење IVI surface-а који се морао користити због компатибилности са Martini и Yocto пројектима.

```
struct wl_surface *surface = wl_compositor_create_surface(compositor);
if (surface == NULL) {
    fprintf(stderr, "Can't create surface\n");
    exit(1);
} else {
    fprintf(stderr, "Created surface\n");
}

if (ivi_application != NULL)
{
    surface_i = ivi_application_surface_create(ivi_application, id_ivisurf, surface);
    if (surface_i == NULL)
    {
        fprintf(stderr, "Cant create ivi surface");
        return 1;
    }
}

if (ivi_Wm != NULL)
{
    ivi_wm_layer_add_surface(iviWm, layerId, surfaceId);

    ivi_wm_commit_changes(iviWm);
}
```

Слика 13 – Креирање и конфигурација површине у Wayland пројекту

Будући да се Martini програмско решење развија за системе у возилима, логично је да ће се користити IVI систем. Помоћу функције ivi_application_surface_create() се прави нова површина на оцнову обичне Wayland површине. Затим се додаје у одговарајући слој и примењују се све промене, помоћу ivi_Wm.

4.3 Обрада улаза и мапирање тастера

Главна функционалност овог Wayland клијента је управо обрада улаза са тастатуре и прослеђивање у Андроид. Руковање догађајима који се односе на тастатуру је имплементирано помоћу слушаоца, слично као код руковања глобалним објектима. Коришћена је и xkbcommon библиотека за управљање распоредом и стањем тастера.

```
static const struct wl_keyboard_listener keyboard_listener = {
    .keymap = keyboard_handle_keymap,
    .enter = keyboard_enter_handler,
    .leave = keyboard_leave_handle,
    .key = keyboard_handle_key,
    .modifiers = keyboard_handle_modifiers
};
```

Слика 14 – Приказ слушаоца догађаја тастатуре

Даља имплементација подразумева дефинисање функција везаних за тастатуру. У keyboard_handle_key() функцији, осим мапирања, било је потребно и проследити дати карактер помоћу функције send() која је део UDS протокола.

```c
void keyboard_handle_key(void *data, struct wl_keyboard *keyboard, uint32_t serial, uint32_t time, uint32_t key, uint32_t state)
{
    if (state == WL_KEYBOARD_KEY_STATE_PRESSED) {
        // Get the Unicode representation of the pressed key
        xkb_keysym_t sym = xkb_state_key_get_one_sym(xkb_state, key + 8);
        if (sym != XKB_KEY_NoSymbol) {
            // Get the Unicode character associated with the keysym
            xkb_layout_index_t group = xkb_state_key_get_layout(xkb_state, key + 8);
            xkb_keycode_t keycode = key + 8; // Convert to xkb keycode
            char buff[8]; // Maximum size of UTF-8 character is 4 bytes
            if (xkb_state_key_get_utf8(xkb_state, keycode, buff, sizeof(buff)) > 0) {
                printf("Pressed key: %s\n", buff);

                printf("Message to server: ");
                fflush(stdout);

                // Send the pressed key to the client
                send(client_fd, buff, strlen(buff), 0);
            } else {
                printf("Failed to get UTF-8 representation of pressed key\n");
            }
        } else {
            printf("Pressed key does not have a Unicode representation\n");
        }
    }
}
```

Слика 15 – Обрада, мапирање и слање унешеног карактера

4.4 Имплементација UDS сервера и клијента

Комуникација између два процеса се у овом задатку обавља преко утичнице. Ова врста утичнице (Unix Domain Socket) је изабрана због тога што оба процеса имају приступ истом кернелу, иако се један налази на домаћину, а други у контејнеру. Комуникација се не одвија путем мреже већ коришћењем система датотека за управљање утичницама. Ово омогућава бољу контролу над тиме ко може приступити одређеној утичници. Наравно, приликом имплементације је због овога потребно водити рачуна о правима приступа.

Као што је већ раније напоменуто, комуникација се заснива на клијент-сервер односу, где је клијент интегрисан у сервис за управљање тастатуром, док се сервер налази у Андроиду. Сервер је имплементиран у оквиру Hwcomposer-a који представља кључну компоненту која омогућава Андроид оперативном систему да комуницира са хардверским подсистемом за приказ. Подаци који пристижу од стране клијента се приказују на екран помоћу функција које се налазе у овој датотеци, што је и главни разлог интегрисања сервера на баш овој локацији у систему. Као што се може видети на следећој слици, сервер је тај који прави утичницу преко које ће се одвијати комуникација. Он затим повезује утичницу са одређеном путањом, слуша везе и прихвата их.

```
void mARTiniHWC2::Display::socketConnect() {

    server_fd = socket(AF_UNIX, SOCK_STREAM, 0);
    if (server_fd == -1) {
        ALOGE("Socket creation failed from socketConnect");
        return;
    }

    memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sun_family = AF_UNIX;
    strncpy(server_addr.sun_path, SOCKET_PATH, sizeof(server_addr.sun_path) - 1);

    unlink(SOCKET_PATH);
    if (bind(server_fd, (struct sockaddr*)&server_addr, sizeof(server_addr)) == -1) {
        ALOGE("Bind failed from socketConnect: %s", strerror(errno));
        close(server_fd);
        return;
    }

    if (listen(server_fd, 1) == -1) {
        ALOGE("Listen failed from socketConnect");
        close(server_fd);
        return;
    }

    ALOGI("Server is listening from socketConnect...");

    client_len = sizeof(client_addr);
    client_fd = accept(server_fd, (struct sockaddr*)&client_addr, &client_len);
    if (client_fd == -1) {
        ALOGE("Accept failed from socketConnect");
        close(server_fd);
        return;
    }

    ALOGI("Connection established! from socketConnect");

    char response[1024];
    strncpy(response, "Start sending data to me now!\n", sizeof(response));
    send(client_fd, response, strlen(response), 0);


}
```

Слика 16 – Део имплементације UDS сервера

Сервер такође шаље поруку клијенту да је спреман за пријем података. Функција socketConnect() се позива сваки пут када се поље за унос текста стави у фокус.

Примање података је имплементирано у оквиру функције која шаље те податке на екран. На крају, када више није потребно да постоји веза између клијента и сервера, утичница се мора затворити.

```
void
mARTiniHWC2::Display::keyboardKey([[maybe_unused]] void *data, [[maybe_unused]] struct wl_keyboard *wl_keyboard,
                                  [[maybe_unused]] uint32_t serial, [[maybe_unused]] uint32_t time,
                                  [[maybe_unused]] uint32_t key,
                                  [[maybe_unused]] uint32_t state) {
    auto *display = static_cast<Display *> (data);

    memset(buffer, 0, sizeof(buffer));
    ssize_t bytes_received = recv(client_fd, buffer, sizeof(buffer), 0);
    if (bytes_received == -1) {
        ALOGE("Receive failed from keyboardKey");
    } else if (bytes_received == 0) {
        ALOGI("Client disconnected from keyboardKey");
    }

    int res = write(display->mInputFd, buffer, sizeof(buffer));
    if (res < sizeof(buffer)) {
        ALOGE("Failed to write keyboard event");
    }
}
```

Слика 17 – Примање података од стране сервера и слање на екран

Са друге стране, клијент успоставља везу са сервером на одговарајућој путањи и прима иницијалну поруку од сервера, која говори о томе да је сервер спреман да прима податке. Слање података је имплементирано у функцији keyboard_handle_key(), која је претходно приказана на слици 15.

```
void socketConnection() {

    client_fd = socket(AF_UNIX, SOCK_STREAM, 0);
    if (client_fd == -1) {
        perror("Socket creation failed.");
        return;
    }

    memset(&client_addr, 0, sizeof(client_addr));
    client_addr.sun_family = AF_UNIX;
    strncpy(client_addr.sun_path, SOCKET_PATH, sizeof(client_addr.sun_path) - 1);

    if (connect(client_fd, (struct sockaddr*)&client_addr, sizeof(client_addr)) == -1) {
        perror("Connect failed.");
        close(client_fd);
        return;
    }

    memset(buffer, 0, sizeof(buffer));
    ssize_t bytes_received = recv(client_fd, buffer, sizeof(buffer), 0);
    if (bytes_received == -1) {
        perror("Receive failed.");
    } else if (bytes_received == 0) {
        printf("Server disconnected.\n");
    }

    printf("Received from server: %s\n", buffer);

}
```

Слика 18 – Део имплементације UDS клијента

У обе имплементације је било непходно навести путању на којој ће се направити утичница и која ће бити видљива и једној и другој страни. У изради овог задатка, посебно се морало водити рачуна о видљивости директоријума који се користе и од стране клијента и сервера. Контејнери имају свој изоловани систем датотека, одвојен од домаћина, што значи да немају приступ фајловима изван свог система. Будући да се клијент налази на Qemu(што ће бити детаљније описано у следећем поднаслову), а сервер у Андроиду, било је неопходно користити технику прикључивања. У конфигурацију која се укључује приликом покретања контејнера, додата је команда lxc.mount.entry којој је речено са које локације и на коју локацију се прикључује одговарајући директоријум. Коришћењем ове опције, утичница може без проблема бити коришћена у оба смера.

4.5 Пребацивање сервиса на Qemu

Да би направљени сервис (Wayland клијент) могао да се извршава у оквиру Yocto пројекта, најпре је било потребно направити CMake за превођење имплементираног сервиса. Коришћењем овог алата се заправо постиже аутоматизација изградње софтвера и генеришу се потребне скрипте и командне датотеке за превођење на жељеној платформи (генеришу се Makefile-ови).

```
cmake_minimum_required(VERSION 3.10)
project(keyboardService)

#set standard for c
set(CMAKE_C_STANDARD 11)

add_executable(keyboardService main2.c)

target_include_directories(keyboardService PRIVATE
        "include"
        "/home/bjana/Projects/martini_manifest/yocto-build/qemu/yocto-x86/build/tmp/work/wayland-ivi-extension/build/protocol"
        "/home/bjana/Projects/martini_manifest/yocto-build/qemu/yocto-x86/build/tmp/work/wayland-ivi-extension/build/ivi-layermanagement-api/ilmControl"
)


add_library(ivi-controller SHARED IMPORTED)
add_library(exec_weston SHARED IMPORTED)

set_target_properties(ivi-controller PROPERTIES IMPORTED_LOCATION /home/bjana/Projects/martini_manifest/yocto-build/qemu/yocto-x86/build/tmp/work/ivi-controller.so)
set_target_properties(exec_weston PROPERTIES IMPORTED_LOCATION /home/bjana/Projects/martini_manifest/yocto-build/qemu/yocto-x86/build/tmp/work/libexec_weston.so)

target_link_libraries(keyboardService
        wayland-client
        ivi-controller
        ivi-application
        exec_weston
        xkbcommon)
```

Слика 19 – Имплементација CMake.txt датотеке

Након овог корака, CMake.txt и main2.c (у којој је имплементиран сервис) датотеке су копиране(премештене) у Martini програм. Даље је било неопходно написати рецепт за покретање овог сервиса, будући да рецепти представљају један од кључних елемената који омогућавају изградњу софтвера за уграђене системе. Они описују како се софтвер преузима, компајлира, пакује и инсталира у циљни систем. Обично је то датотека која се завршава са .bb екстензијом и садржи упутства која BitBake алат користи за изградњу.

У рецепту који је приказан на слици 20, најпре је дат кратак опис, а затим је дефинисана локација изворних датотека. Након су наведени сви пакети од којих овај рецепт зависи,а онда је дефинисана и променљива која чува путању до директоријума са фајловима који су потребни за изградњу. Додатно су укључене још неке опције за CMake, као и опције за паковање. Функција која дефинише како ће се изграђени бинарни фајлови инсталирати је do_install(). Она такође креира и директоријум за бинарне фајлове и копира keyboardService у тај директоријум.

```
SUMMARY = "Recipe for wayland-keyboard-client"
DESCRIPTION = "Recipe created for running test application"
SECTION = "martini"
LICENSE = "CLOSED"

PR = "r0"

# GITUSER ?= "${USER}"
# SRCREV = "52389891a170e92957251d61f12abb617de6bc3d"

# S="${WORKDIR}/${BPN}-${PV}"

SRC_URI = "file://CMakeLists.txt    \
           file://main2.c           \
          "


DEPENDS += "weston wayland-native libxkbcommon wayland wayland-protocols libinput virtual/egl pango wayland-ivi-extension"
SRC_FILES="${@os.path.dirname(d.getVar('FILE', True))}/files"
inherit cmake

EXTRA_OECMAKE += "-DCMAKE_SKIP_RPATH=TRUE"
OECMAKE_GENERATOR = "Unix Makefiles"

INSANE_SKIP_${PN} = "ldflags"
INHIBIT_PACKAGE_STRIP = "1"
INHIBIT_SYSROOT_STRIP = "1"
SOLIBS = ".so"
FILES_SOLIBSDEV = ""

do_unpack() {
    cp ${SRC_FILES}/CMakeLists.txt ${S}
    cp ${SRC_FILES}/main2.c        ${S}
}

do_install() {
    install -d ${D}${bindir}
    install -m 0755 ${B}/keyboardService ${D}${bindir}/
}
```

Слика 20 – Рецепт за изградњу сервиса за тастатуру

Када је програм успешно преведен заједно са рецептом, сервис је могуће покренути у оквиру Qemu терминала једноставним уношењем имена сервиса – keyboardService.

4.6 Захтев за коришћење сервиса

Како би се управљање виртуалном тастатуром у потпуности преусмерило на сервис који је изван Андроида, најпре је било потребно лоцирати методу која прима захтев за позивање виртуалне тастатуре, односно која се позива када је потребно.

У датотеци InputMethodManager.java налази се метода под називом showSoftInput() која се позива сваки пут када постоји потреба за коришћењем виртуалне тастатуре. Идеја је била да се из те методе пошаље порука или сигнал ка сервису на домаћину, који ће по пријему тог сигнала да активира своју функционалност. Реч сигнал се у овом случају односи на назнаку, наговештај.

Ова идеја је реализована помоћу већ постојеће UDS утичнице. Прво се из датотеке InputMethodManager.java шаље захтев у облику карактера ка серверу, који је имплементиран у датотеци hwcomposer.cpp. Затим се овај захтев прослеђује даље ка сервису за управљање виртуалном тастатуром, коришћењем већ успостављене комуникације. Пријемом захтева у сервису омогућује се даља комуникација са Андроидом, у виду слања обрађених улаза са тастатуре.

```java
private LocalSocket socket;

// Adding method for sending data over already existing Unix Domain Socket

public void sendDataOverUnixSocket() {
    try {
        String socketPath = "/data/run/unix-socket";

        Log.d(TAG, "Connecting to Unix socket\n");
        LocalSocketAddress mSocketPath = new LocalSocketAddress(socketPath, LocalSocketAddress.Namespace.FILESYSTEM);
        socket = new LocalSocket();

        socket.connect(mSocketPath);
        Log.d(TAG, "Connected to Unix socket");

        OutputStream outputStream = socket.getOutputStream();

        char data = 'a';

        // Convert char to byte array with UTF-8 encoding
        String charString = Character.toString(data);
        byte[] charBytes = charString.getBytes(StandardCharsets.UTF_8);
        Log.d(TAG, "Sending Unix data: " + charString);

        outputStream.write(charBytes);
        outputStream.flush();
        Log.d(TAG, "Unix Data sent successfully");

        outputStream.close();
        socket.close();
        Log.d(TAG, "Unix Socket and streams closed");
    } catch (Exception e) {
        Log.e(TAG, "Error sending data over Unix socket", e);
        e.printStackTrace();
    }
}
```

Слика 21 – Слање захтева из датотеке InputMethodManager.java

На слици изнад имплементиран је још један UDS клијент, али овај пут унутар Андроид радног оквира. На слици испод приказана је метода showSoftInput() где се и врши позивање функције за слање захтева.

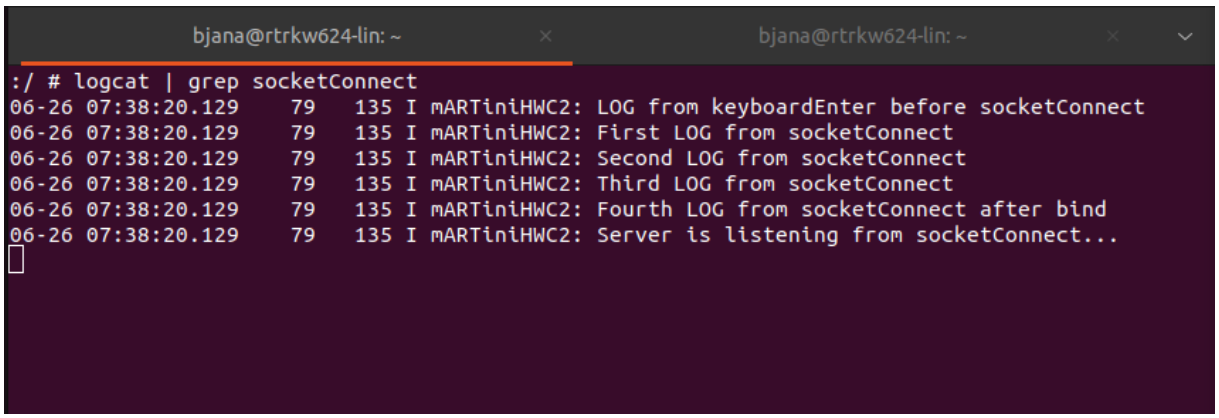
```java
/**
 * Synonym for {@link #showSoftInput(View, int, ResultReceiver)} without
 * a result receiver: explicitly request that the current input method's
 * soft input area be shown to the user, if needed.
 *
 * @param view The currently focused view, which would like to receive
 * soft keyboard input.
 * @param flags Provides additional operating flags.  Currently may be
 * 0 or have the {@link #SHOW_IMPLICIT} bit set.
 */
public boolean showSoftInput(View view, int flags) {
    // Re-dispatch if there is a context mismatch.
    final InputMethodManager fallbackImm = getFallbackInputMethodManagerIfNecessary(view);
    if (fallbackImm != null) {
        return fallbackImm.showSoftInput(view, flags);
    }

    sendDataOverUnixSocket();

    Log.d(TAG, "JANA'S LOG from InputMethodManager::showSoftInput() Unix");

    return showSoftInput(view, flags, null);
}
```

Слика 22 – Приказ методе showSoftInput()

5. Резултати

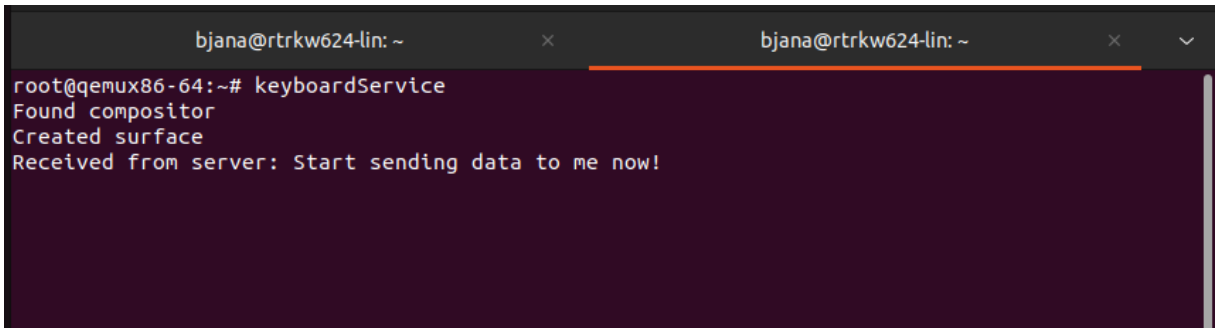
Резултати овог рада се првенствено огледају у успешној комуникацији између сервиса који се налази на Линукс домаћину и Андроид оперативног система који се користи у Martini infotainment систему, а затим и у успешној обради и преносу улазних карактера са тастатуре. Имплементација и функционалност решења су тестирани покретањем самог сервиса, што је уједно и забележено снимцима екрана. Такође, резултати се највећим делом приказују кроз исписе и логове у терминалу, јер се имплементирана функционалност највећим делом одвија у позадини, што кориснику није видљиво.

На следећој слици су приказани логови који се појављују на Андроид страни (на страни UDS сервера) када се притисне на поље за унос текста, тј. када се појави потреба за коришћењем виртуалне тастатуре.



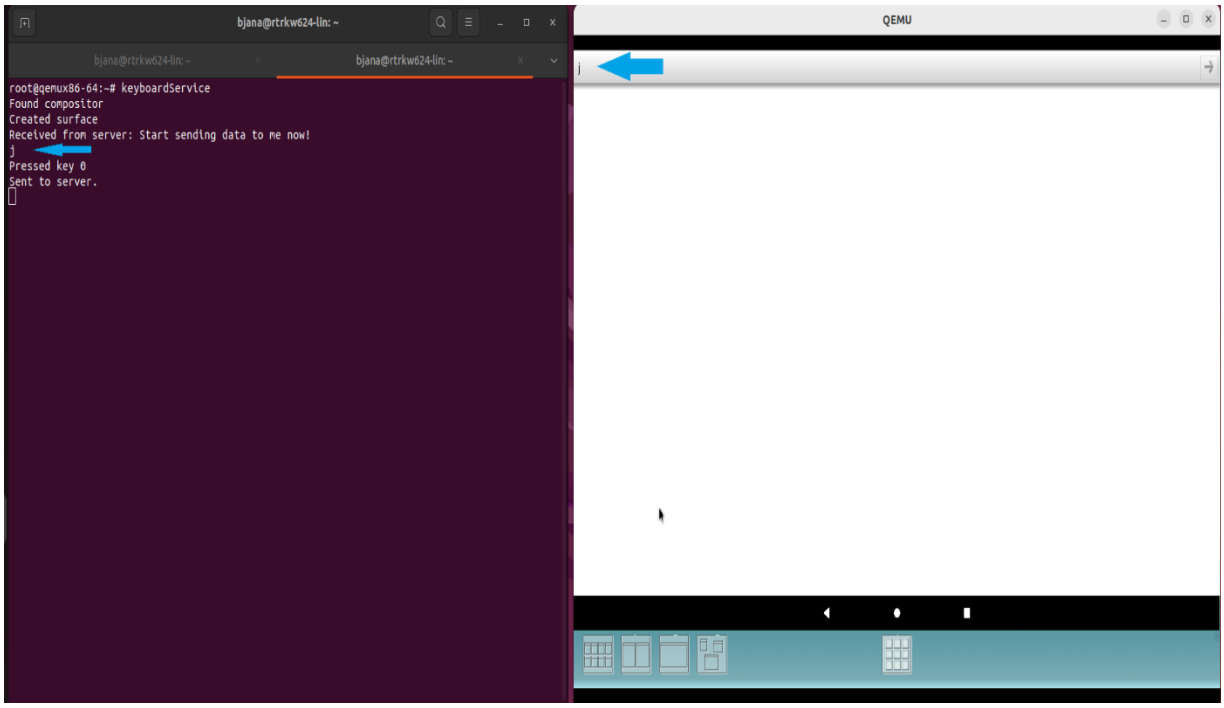
Слика 23 - Приказ серверских логова након притискања поља за унос текста

Тада се уједно и позове метода showSoftInput() која шаље такозвани захтев за покретање сервиса за тастатуру. По пријему тог захтева, сервис прелази у стање из ког може да шаље податке ка Андроиду.



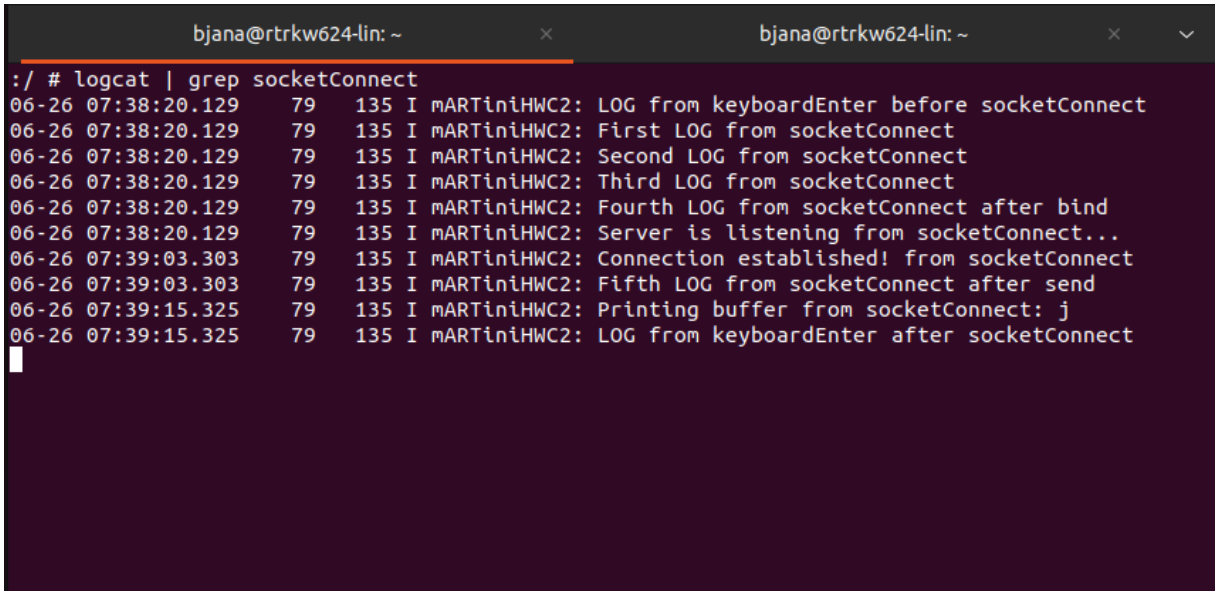
Слика 24 – Исписи на страни клијента након успешног повезивања са сервером и пријема захтева за коришћење виртуалне тастатуре

Клијент затим шаље обрађене карактере које је корисник унео, који се на крају и приказују у пољу за унос текста.



Слика 25 – Успешно слање и приказ карактера у пољу за унос текста

Успешна комуникација и пријем карактера се може проверити и на серверској страни UDS протокола.



Слика 26 – Приказ логова на серверској страни након успешног пријема карактера

6. Закључак

Израда овог задатка захтевала је детаљно истраживање и имплементацију Wayland програмског клијента за управљање виртуалном тастатуром у оквиру Андроид оперативног система који се извршава у LXC контејнеру на Линукс домаћину. Коришћењем Yocto пројекта, који је омогућио креирање у прилагођеној Линукс дистрибуцији, успешно је остварена интеграција и контрола виртуалне тастатуре.

Процес имплементације је захтевао дубоко разумевање како Wayland протокола, тако и интеракције између различитих софтверских слојева у уграђеним системима. Успешно повезивање теоријских основа са практичном применом, резултирало је функционалним и ефикасним решењем које потенцијално може да допринесе унапређењу корисничког искуства у модерним аутомобилским системима.

Главни резултати овог рада показују да је могуће значајно унапредити функционалност у аутомобилским infotainment системима путем увођења нових технологија и приступа. Поред тога, овај рад отвара нове могућности за даља истраживања у области интеграције софтвера за уграђене системе, посебно у контексту аутомобилске индустрије која се брзо развија.

7. Литература

[1] Martini architecture, https://www.rt-rk.com/products/martini/, посећено 06.06.2024.

[2] Android characteristics, https://source.android.com/, посећено 07.06.2024.

[3] Android architecture, https://source.android.com/docs/core/architecture, посећено 07.06.2024.

[4] Wayland architecture, https://wayland.freedesktop.org/architecture.html, посећено 15.06.2024.

[5] UDS communication channel, https://medium.com/swlh/getting-started-with-unix-domain-sockets-4472c0db4eb1, посећено 19.06.2024.

[6] In-Vehicle Infotainment, https://www.autopi.io/glossary/in-vehicle-infotainment/, посећено 20.06.2024.