FELADATKIÍRÁS

Az elektronikusan beadott változatban ez az oldal törlendő. A nyomtatott változatban ennek az oldalnak a helyére a diplomaterv portálról letöltött, jóváhagyott feladatkiírást kell befűzni.

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Sándor Dávid

BIOMETRIKUS AUTENTIKÁCIÓS MEGOLDÁS FEJLESZTÉSE

KONZULENS

Dr. Kővári Bence

Golda Bence

BUDAPEST, 2022

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott **Sándor Dávid**, szigorló hallgató kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző, cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2022. 04. 21

...…….
Sándor Dávid

Összefoglaló

A szakdolgozat, vagy diplomaterv elkészítése minden egyetemi hallgató életében egy fontos mérföldkő. Lehetőséget ad arra, hogy az egyetemi évei során megtanultakat kamatoztassa és eredményeit szélesebb közönség előtt bemutassa, s mérnöki rátermettségét bizonyítsa. Fontos azonban, hogy a dolgozat elkészítésének folyamata számos csapdát is rejt magában. Rossz időgazdálkodás, hiányos szövegszerkesztési ismeretek, illetve a dolgozat készítéséhez nélkülözhetetlen „műfaji" szabályok ismeretének hiánya könnyen oda vezethetnek, hogy egy egyébként jelentős időbefektetéssel készült kiemelkedő szoftver is csak gyengébb minősítést kapjon a gyenge minőségű dolgozat miatt.

E dokumentum – amellett, hogy egy általános szerkesztési keretet ad a dolgozatodnak – összefoglalja a szakdolgozat/diplomaterv írás írott és íratlan szabályait. Összeszedjük a Word kezelésének legfontosabb részeit (címsorok, ábrák, irodalomjegyzék stb.), a dolgozat felépítésének általános tartalmi és szerkezeti irányelveit. Bár mindenkire igazítható sablon természetesen nem létezik, megadjuk azokat az általános arányokat, oldalszámokat, amelyek betartásával jó eséllyel készíthetsz egy színvonalas dolgozatot. A részletes és pontokba szedett elvárás-lista nem csupán a dolgozat írásakor, de akár más dolgozatok értékelésekor is kiváló támpontként szolgálhat.

Az itt átadott ismeretek és szemléletmód nem csupán az aktuális feladatod leküzdésében segíthet, de hosszútávon is számos praktikus fogással bővítheti a szövegszerkesztési és dokumentumkészítési eszköztáradat.

Abstract

Ide jön a ½-1 oldalas angol nyelvű összefoglaló, amelynek szövege a Diplomaterv Portálra külön is feltöltésre kerül. Ez a magyar nyelvű összefoglaló angolra fordított változata.

1 Bevezetés

A feladat bemutatása.

1.1 Motiváció

A motiváció ismertetése.

1.2 A dolgozat szerkezete

A dolgozat szerkezetének bemutatása.

2 Háttérismeretek

Az alábbi fejezetben olyan témakörök, fogalmak, technológiák kerülnek bemutatásra, amelyek a diplomaterv értelmezését segítik. A fejezetnek nem célja az adott témakörök részletes dokumentációja, a hangsúly minden esetben a diplomaterv megértéséhez elengedhetetlenül fontos fogalmak bemutatásán van. Amennyiben az Olvasó egy adott témakörhöz kapcsolódó további szakirodalmat keres, ajánlom az irodalomjegyzékben összegyűjtött források, hivatkozások olvasását.

2.1 Megjelenítő protokollok [1]

A Unix-szerű operációs rendszerek esetén a grafikus felhasználói felület (amennyiben a rendszer egyáltalán rendelkezik GUI-val (Graphical User Interface)) meglehetősen összetett. Egy GUI általában több különböző komponensből áll, például (a teljeség igénye nélkül): ablakozó szoftver, widget könyvtárak, bemeneti / kimeneti eszközök, stb. A helyzetet tovább bonyolítja, hogy eltérő Linux disztribúciókon más és más implementációkkal találkozhatunk, amelyek gyakran nem kompatibilisek egymással.

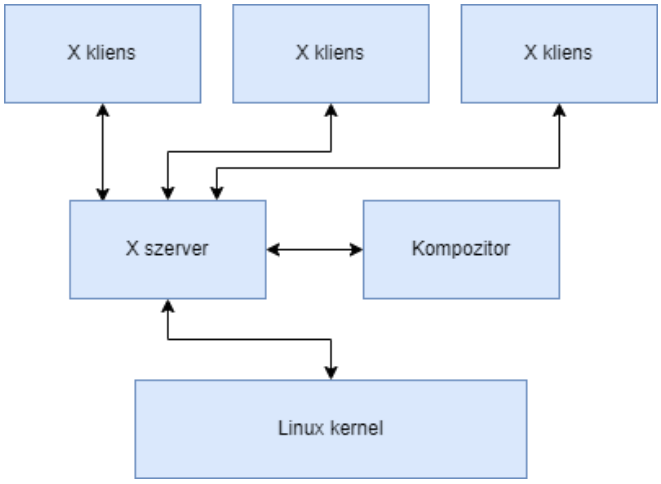
Azt a szoftvert, ami a különböző GUI komponenseket összefogja és lehetővé teszi, hogy ezek hatékonyan együttműködjenek megjelenítő szervernek (display server) hívják. A megjelenítő szerver kezeli az alsóbb szintű funkciókat, közvetlenül kommunikál a kernellel (ezen keresztül pedig a hardver erőforrásokkal). A képernyőre való rajzolást és a bemeneti / kimeneti eszközök adatainak továbbítását a grafikus alkalmazások felé is a megjelenítő szerver végzi. A többi felsőbb szintű komponenst integrálja, interfészeket biztosít, amelyeken keresztül az alsóbb szintű funkciók elérhetővé válnak. Fontos megjegyezni a különbséget a megjelenítő szerver és az asztali környezet (desktop environment) között. A legtöbb grafikus interfésszel rendelkező Linux disztribúció valamilyen asztali környezetet használ (GNOME, KDE, Xfce, stb.). Ezek a szoftverek különböző GUI elemeket biztosítanak (ikonok, widget-ek, háttérképek), interfészeket nyújtanak grafikus felületek programozására. Architekturálisan egy magasabb absztrakciós szinten helyezkednek el, mint a megjelenítő szerver, nem kommunikálnak közvetlenül a kernellel.

A megjelenítő szerver kliensének tekintünk általában minden grafikus felülettel rendelkező alkalmazást, de természetesen GUI nélküli programok is lehetnek kliens alkalmazások. A megjelenítő szerver a klienseivel a megjelenítő protokollon (display protocol) keresztül kommunikál. Linux rendszerek esetén többféle megjelenítő protokollal találkozhatunk és egy adott protokollhoz általában többféle implementáció is létezik. Az idők során két megjelenítő protokoll terjedt el nagyobb körben, az X Window System, illetve a Wayland protocoll. Mivel a legnépszerűbb grafikus interfésszel rendelkező Linux disztribúciók LTS (Long Term Support) verziója szinte kivétel nélkül a fent említett megjelenítő protokollok egyikét használja, ezért a diplomaterv során további megjelenítő protokollokkal nem foglalkoztam.

2.1.1 X Window System

Az X Window System (X11, helyenként csak X) egy nyílt forráskódú megjelenítő protokoll készlet, amely Unix-szerű rendszerekhez készült. A protokollt 1984-ben kezdték el kifejleszteni és jelenleg a 11-es verziónál tart (innen ered az X11 kifejezés). Maga a protokoll egy szöveges leírás, ami publikusan elérhető. A protokollhoz tartozik egy szerveroldali referencia implementáció, amit X.Org Server-nek hívnak. A protokollt megvalósító elterjedtebb C-ben implementált kliensoldali könyvtárak az Xlib és az XCB.

Architektúráját tekintve az X Window System egy kliens-szerver architektúrát valósít meg. Az X szerver vezérli a fizikai megjelenítő készülékeket és feldolgozza a bemeneti eszközöktől érkező adatokat. Az X kliensek olyan alkalmazások, amelyek az X szerveren keresztül szeretnének interakcióba lépni a bemeneti / kimeneti eszközökkel.



1. ábra Az X Window System architektúrája

A rendszerhez tartozik még egy komponens, a kompozitor. A kompozitor feladata, hogy a különböző ablakok elrendezését vezérelje a képernyőn.

Bizonyos kifejezéseket az X Window System árnyaltabban használ a közbeszédhez képest, a legfontosabbak ezek közül a következők:

- **device (eszköz)** – Dedikált vagy alaplapra integrált videókártya.
- **monitor** – Fizikai megjelenítő eszköz.
- **screen (képernyő)** – Egy olyan terület, amelyre grafikus tartalmat lehet renderelni. Ez egyszerre több monitoron is megjelenhet (akár duplikálva, akár kiterjesztve).
- **display (kijelző)** – Képernyők gyűjteménye, amely gyakran több monitort foglal magába. A Linux-alapú számítógépek általában képesek arra, hogy több kijelzővel rendelkezzenek egyidejűleg. Ezek között a felhasználó egy speciális billentyűkombinációval, például a control-alt-funkcióbillentyűvel válthat, átkapcsolva az összes monitort az egyik kijelző képernyőinek megjelenítéséről a másik kijelző képernyőire.

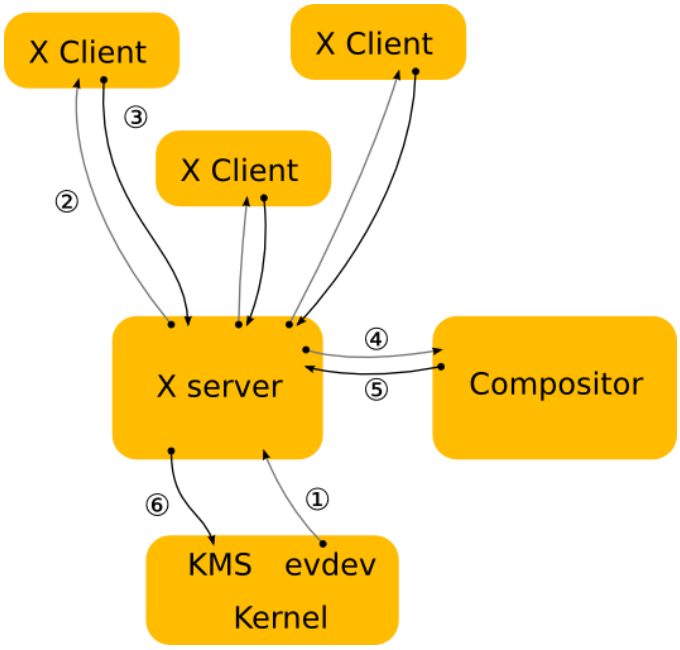
Az X protokoll négy különböző üzenet típust definiál, amelyeknek a szerver és a kliensek közti kommunikációban van szerepe. A protokoll a következő üzenet típusokat különbözteti meg:

- **request** – A kliens küldi a szervernek. Egy request sokféle információt tartalmazhat, mint például egy új ablak létehozását, vagy a kurzor pozíciójának lekérdezését.
- **reply** – A szerver küldi a kliensek. A reply üzenetek a request üzenetek hatására jönnek létre és a kliens által kért információt tartalmazzák.
- **event** – A szerver küldi a kliensnek. Az ilyen típusú üzeneteket általában nem közvetlenül a kliens váltja ki. Sokféle típusú event üzenet létezik, ilyen például a bemeneti eszközök (például billentyűzet vagy egér) által generált események.
- **error** – A szerver küldi a kliensnek. Hasonlóan működnek az event típusú üzenetekhez, valamilyen hiba fennállását jelzik.

2.1.2 Wayland

A Wayland egy ingyenes, nyílt forráskódú megjelenítő protokoll. A Wayland projekt célja, hogy leváltsa az X Window System-et egy modernebb, egyszerűbb és biztonságosabb protokollra. A protokollt 2008-ban kezdték el fejleszteni és a mai napig aktívan dolgoznak rajta. Az X Window System-hez hasonlóan a Wayland protokoll is egy szerver-kliens architektúrát követ. Ellentétben az X-szel a Wayland esetében a megjelenítő szervert kompozitornak hívják. Ez abból az alapvető architektúrális különbségből ered, hogy a Wayland esetében a megjelenítő szerver és a kompozitor egy komponensként funkcionál. A protokollhoz tartozik egy szerveroldali referencia implementáció C-ben, amit Weston kompozitornak hívnak. A legnépszerűbb kliensoldali könyvtár a libwayland szintén C-ben lett implementálva.

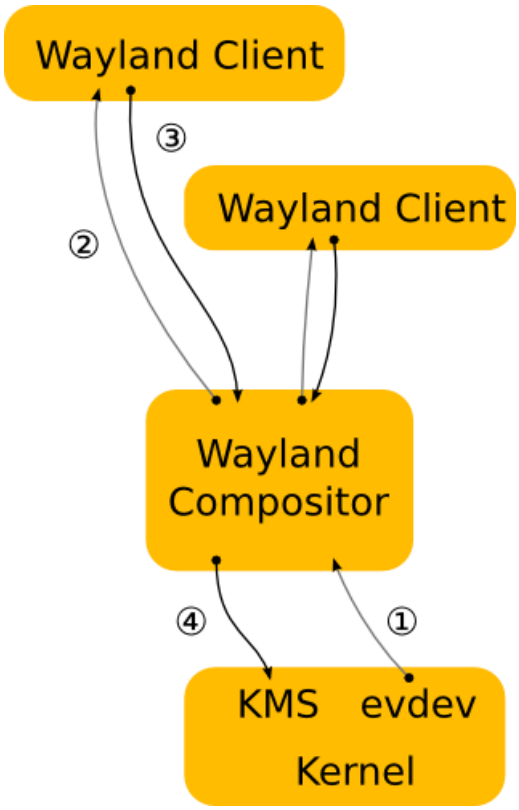
A legjobb módja annak, hogy összehasonlítsuk a Wayland és az X Window System architektúráját és megértsük a különbségeket az, ha végig követjük egy bemeneti eszköz által generált esemény útját egészen addig, ameddig az esemény által kiváltott változás megjelenik a képernyőn. Az X esetében ez a következőképpen zajlik le:



2. ábra Egy esemény feldolgozása az X-szen.

1. A kernel kap egy eseményt egy bemeneti eszköztől és elküldi az X szervernek a bemeneti vezérlőn (evdev driver) keresztül. A különböző eszközspecifikus eseményprotokollok lefordítását az evdev szabványra a kernel végzi el.
2. Az X szerver meghatározza, hogy melyik ablakot érintette az esemény és elküldi azoknak az X klienseknek, amelyek az adott ablakban a kérdéses eseményre feliratkoztak.
3. A kliensek feldolgozzák az eseményt és eldöntik, hogy mit tegyenek. Sokszor egy bemeneti esemény hatására a felhasználói felületnek meg kell változnia, például a felhasználó egy hivatkozás felé viszi a kurzort, vagy bepipál egy checkbox-ot. Az esemény feldolgozása után a kliens egy renderelési kérést (request típusú üzenet) küld a szervernek.
4. Az X szerver megkapja a renderelési kérést és egy illesztőprogramon keresztül szól a hardvernek, hogy az végezze el a renderelést. Az X szerver továbbá kiszámítja a renderelés határoló régióját és elküldi ezt a kompozitornak egy káreseménynek (damage event) nevezett üzenetben. Erre azért van szükség, mert a kompozitornak a bemeneti esemény hatására lehet, hogy bizonyos effekteket kell alkalmaznia (forgatás, skálázás, stb).
5. A káreseményből a kompozitor megtudja, hogy valami megváltozott az ablakban és az képernyőnek azt a részét újra kell komponálnia, ahol az ablak megváltozott. Miután ezt megtette, a kompozitor küld egy renderelési kérést az X szervernek.
6. Az X szerver megkapja a renderelési kérést a kompozitortól és végrehajtja azt.

A Wayland protokoll esetén a kompozitor és a megjlenítő szerver egy és ugyanaz a komponens. Ez lehetővé teszi, hogy a kompozitor közvetlenül a klienseknek küldje a bemeneti eseményeket és fordítva a kliensek közvetlenül a kompozitornak küldik a káreseményeket. Ugyanez a folyamat a Wayland esetében a következőképpen zajlik le:



3. ábra Egy esemény feldolgozása Wayland-en.

1. A kernel kap egy eseményt egy bemeneti eszköztől és elküldi a Wayland kompozitornak a bemeneti vezérlőn keresztül.
2. A kompozitor meghatározza, hogy melyik ablakot érintette az adottt esemény és tájékoztatja erről az érintett klienseket. A kompozitor érti a különböző effekteket, transzformációkat, amikkel az egyes elemek rendelkezhetnek, így képes az ablak-lokális – képernyő-lokális koordináták fordítására. Ezáltal a kompozitor pontosan meg tudja határozni, hogy melyik ablakot érintette az adott esemény, feleslegessé válik az X-es acrcitektúrában a 4-es és az 5-ös lépés.
3. Az X-es architektúrához hasonlóan a kliens megkapja az eseményt és feldolgozza azt. Egy újabb különbség a protokollok között, hogy a Wayland esetén a renderelés kliens oldalon történik (gyakorlatban a kliens a kompozitorral megosztott közös videómemória pufferbe

renderel). Végezetül a kliens értesíti a kompozitort, hogy jelezze, hogy a felhasználói felületen változás történt (káresemény).

4. A kompozitor összegyűjti a kliensektől a káreseményeket és újra összeállítja a képernyőt.

Az X Window System architektúrájában a kompozitor felelős azért, hogy mindent megjelenítsen a képernyőn, de ezt mégis az X szerveren keresztül kell tennie. Lényegében az X szerver egy közvetítő szerepet játszik a kliensek és a kompozitor, illetve a kompozitor és a hardver között. A Wayland protokollban azáltal, hogy a meglejelenítő szerver helyére lép a kompozitor lényegesen csökkent a rendszer komplexitása, illetve a kommunikációs többlet.

2.2 Rust

2.3 Erlang

Az Erlang egy univerzális, konkurens, funkcionális programozási nyelv és futási időben *garbage collection* mechanizmussal ellátott környezet. Az Erlang és az Erlang/OTP (Open Telecom Platform) kifejezést sokszor felcserélhető módon használják. Az Erlang/OTP az Erlang környezetből, számos „off-the-shelf” Erlang könyvtárból és tervezési mintából (viselkedésleíró sablon) áll. Az Erlangot a következő jellemvonásokkal rendelkező rendszerek megalkotására tervezték:

- **Elosztottság**: A nyelv magas szinten támogatja a moduláris és konkurrens programozást.
- **Hibatűrés**: Az Erlang számos eszközt és tervezési irányelvet („Let-it-crash”, felügyeleti fa) biztosít, amellyel a hibák előfordulása és a rendszerre gyakorolt hatása minimalizálható.
- **Magas rendelkezésre állás**: Az Erlang folyamatok izolációjából következően, ha egy folyamat hibába ütközik és leáll, az a rendszernek csak egy kisebb, elkülönített részében fog szolgáltatás kiesést okozni.
- **Laza valós idejűség** (Soft real-time): Az Erlangot eredetileg telekommunikációs rendszerek létrehozására alkották meg, így a valós idejű működés egy alapvető kritérium volt a kezdetektől fogva.

- **Kód cserélése futásidőben** (Hot swapping): Az Erlang/OTP-ben található tervezési minták generikus módon támogatják egy futó folyamat kódjának frissítését anélkül, hogy a folyamatot le kéne állítani.

2.3.1 Típusok

A következő leírásban az Erlang nyelv adattípusairól, illetve egyedi típusok specifikálásáról lesz szó. Az Erlang egy dinamikusan erősen típusos (dinamically strongly typed), egyszeri értékadást használó programozási nyelv[11]. Több alap adattípust definiál, ilyenek például az integer, binary vagy az atom. A beépített adattípusok felhasználásával lehetséges saját típusok specifikálása. A típus specifikációnak többek közt dokumentációs célja van, továbbá nagy mértékben növeli a kód olvashatóságát és plusz információval látja el a hiba detektáló eszközöket (például Dialyzer). Egy típus specifikálásának a következő a szintaxisa:

```
-type name() :: datatype()
```

A *-type* kulcsszó jelzi egy modulban, hogy típus specifikáció következik. A name a specifikált típus neve, ezzel a névvel lehet hivatkozni a típusra a modulon belül, a modulon kívül (amennyiben exportálásra kerül a típus) a modulnév:típusnév() szintaxissal lehet hivatkozni a specifikált típusra. A datatype a specifikált típus leírása, ami lehet egy beépített adat típus, egy másik specifikált típus, egy atom vagy integer típusú érték (pl.: „foo” vagy 21), vagy ezek tetszőlegesen vett uniója. Példa egy típus specifikációra:

```
-type mytype() :: boolean() | undefined
```

A *mytype* nevezetű típus ennek értelmében olyan adattípust definiál, amely vagy egy *boolean* értéket vesz fel, vagy az *undefined* atom értékét.

2.3.2 Modulok

Egy Erlang alkalmazásban a kód modulokra van osztva. Egy modulban található kódot két fő részre lehet bontani, a modullal kapcsolatos attribútumok deklarálására (például típus specifikációk vagy exportált függvények listája) és függvény deklarációkra. Gyakran előfordul, hogy két modul strukturálisan nagyon hasonlít egymásra, ugyanazokat a mintákat követik. Ilyenek például a felügyeleti modulok, amelyek általában csak abban térnek el, hogy milyen Erlang folyamatokat felügyelnek. Ezeknek a gyakori strukturális mintáknak a formalizálására létre lehet hozni

úgynevezett viselkedés leíró modulokat. Ezáltal szét lehet választani a kódot egy újrahasználható, generikus részre (viselkedés leíró modul) és egy specifikus részre (callback modul). Az Erlang/OTP-ben vannak előre definiált, beépített viselkedések (például gen_server), de van lehetőségünk saját viselkedés leíró modulokat definiálni[12]. Egy ilyen generikus modulban deklarálhatunk olyan függvényeket, amelyeknek az implementációja kötelező azoknak a callback moduloknak, amelyek megvalósítják ezt a viselkedést. Egy callback függvény deklarációja a következőképpen néz ki:

```
-callback FunctionName(Arg1, Arg2, ..., ArgN) -> Res.
```

A FunctionName a callback függvény neve, az ArgX az X-edik argumentum típusa, a Res pedig az eredmény típusa. Lehetőség van opcionális callback metódusok megadására is, ekkor az -optional_callback kulcsszót kell használni. A viselkedést megvalósító modulban, a következő modul attribútummal tudjuk jelezni az Erlang fordítóprogramjának, hogy ez egy callback modul:

```
-behaviour(Behaviour)
```

A Behaviour Erlang atom egy modul neve kell, hogy legyen. Így a fordítóprogram felismeri, hogy ez egy callback modul és jelezni fogja, ha valamelyik callback metódus nem került implementálásra.

2.4 React

3 Kapcsolódó munka

Ebben a fejezetben az elkészült megoldáshoz vezető kapcsolódó munkámat (irodalomkutatás, prototípusok készítése) fogom bemutatni és a felmerülő tervezői döntéseket megindokolni.

3.1 Wayland prototípus

A natív Linux-on futó adatgyűjtő kliens alkalmazás tervezésekor az egyik alapvető kritérium a minél nagyobb felhasználói csoport támogatása volt. Tekintve a Linux disztribúciók változatos és sokszínű világát ennek a követelménynek korántsem triviális eleget tenni. Ahhoz, hogy minél több disztribúciót (és verziót) támogatni tudjon az alkalmazás, annál kernel-közelebbi szinten kell implementálni a klienst. Valószínűleg nagyban megkönnyítené az implementációt, ha a klienst az asztali környezetek szintjén készíteném el és például GNOME-specifikus lenne. Ugyanakkor ezzel a felhasználóknak egy jelentős hányadát kizárnám, például akik KDE-t vagy Xfce-t használnak. A logikus döntés tehát, hogy egy absztrakciós szinttel alacsonyabban, a megjelenítő szerver szintjén készüljön el az alkalmazás.

A megjelenítő protokollok esetében már nem áll fent a „bőség zavara”, a Wayland protokoll és az X Window System között kell választani. Az X-szet 1984-ben kezdték el fejleszteni és a legújabb nagy verzió kiadása 1987-ben történt (kisebb verzió frissítések jelentek meg, jelenleg a legfrissebb az X11R7.7 2012-ben lett kiadva). Az idők során számos kritika érte az X Window System-et. Leggyakrabban az elavultság, a biztonságtechnikai hiányosságok és a teljesítmény azok a szempontok, amelyek mentén kritikákat fogalmaznak meg a protokollal szemben. Ugyanakkor sok Linux disztribúció a mai napig alapértelmezetten az X.Org-ot használja megjelenítő szervernek és mint opcionális választás szinte mindegyikben megtalálható.

A Wayland protokollt 2008-ban kezdték el fejleszteni (többen az X.org szerver fejlesztő csapatából) és a mai napig aktívan dolgoznak rajta. A projektre „következő generációs” megjelenítő szerverként hivatkoznak és célja, hogy leváltsa az X Window System-et egy modernebb, egyszerűbb és biztonságosabb protokollra. Több nagyobb felhasználóbázissal rendelkező Linux disztribúció (például: Debian, Ubuntu, Fedora) átállt a Wayland-re, mint alapértelmezett megjelenítő protokoll (legalábbis a GNOME

asztali környezetet használó verziók). Mivel a Wayland protokoll modernebb és a jövőben minden bizonnyal át fogja venni az X Window System helyét és az adatgyűjtő kliens alkalmazást alapvetően időtálló módon terveztem implementálni, ezért a Wayland tűnt a jó választásnak.

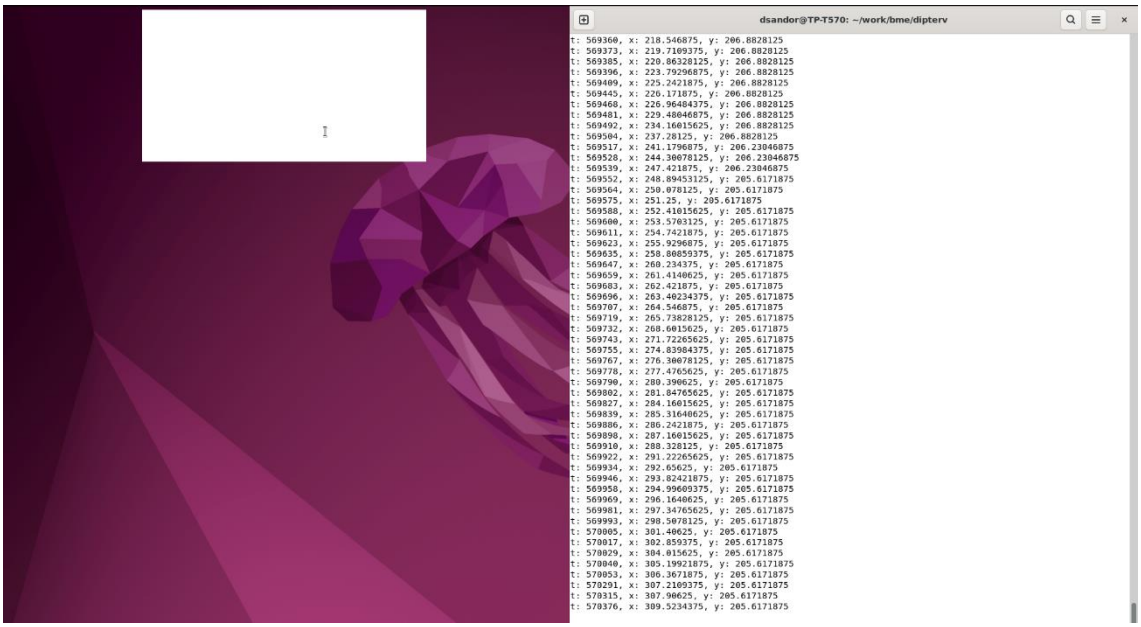
3.1.1 Kutatás és prototípus

Az adatgyűjtő alkalmazást tehát kezdetben egy Wayland kliens alkalmazás formájában próbáltam meg elkészíteni. Első lépésként egy implementációs nyelvet kellett választanom. Ahogy korábban már említettem a Wayland alapvetően egy protokoll, ami XML fájlok formájában van dokumentálva. A protokollhoz tartozik egy szerveroldali referencia implementáció (Weston), illetve egy kliens oldali könyvtár (libwayland), amely C-ben lett implementálva. Kliens oldalon több programozási nyelvhez is készült implementáció (amelyeket túlnyomó részt az XML fájlokból generáltak), illetve a libwayland-hez is elérhető több olyan nyelvi burkoló könyvtár, ami a C-ben implementált függvények hívását teszi lehetővé másik programozási nyelvekből. Mivel a kezdeti célom egy prototípus gyors implementációja volt, ezért a PyWayland-et, egy Python nyelvhez készült burkoló könyvtárat választottam.

A Python-ban implementált prototípus elkészítésével relatíve gyorsan falba ütköztem. Alapvetően azt a konklúziót szűrtem le a prototípus implementálása során, hogy a Wayland protokoll a jelenlegi formájában nem összeegyeztethető az általam elkészíteni kívánt adatgyűjtő alkalmazás követelményeivel. Mielőtt bővebben kifejteném, hogy miért jutottam erre a következtetésre röviden ismertetném a legfontosabb követelményeit az adatgyűjtő kliensnek. Ahogy azt korábban említettem az egyik fontos követelmény a felhasználók minél szélesebb körének támogatása. Ezen kívül a funkcionalitás szempontjából az alkalmazásnak képesnek kell lennie arra, hogy a háttérben, grafikus felhasználó felület nélkül fusson. Továbbá minden egér képességgel rendelkező bemeneti eszköz (egér, touchpad, trackpoint, stb.) által generált kurzor mozgás eseményt fel tudjon dolgozni.

A Wayland protokoll esetén az ablakok az X-hez hasonlóan hierarchikusan helyezkednek el, a tartalmazási fa gyökerében egy speciális ablak található, amit gyökér ablaknak (root window) neveznek. Az egyik központi probléma, amivel találkoztam az volt, hogy a Wayland esetében nincs lehetőség kliens oldalon a gyökér ablak eseményeire feliratkozni. Ennek alapvetően biztonságtechnikai okai vannak, a Wayland

kliensek egymástól izolált környezetben futnak és nem férhetnek hozzá a többi folyamat adataihoz. Egy olyan klienst el tudtam készíteni, ami létrehoz egy alkalmazás ablakot és amikor fókuszba kerül az ablak (a felhasználó az ablak területére mozgatja a kurzort) el kezdi gyűjteni az egér mozgás adatokat.



4. ábra A prototípus kliens működés közben.

Felmerült még ötletként egy teljesképernyős „overlay” alkalmazás ablak készítése, de ez több szempontból sem tűnt jó iránynak. Egyrészt a kurzor mozgás események elkapása csak akkor működik, amikor az ablak fókuszban van, így amikor a felhasználó egy másik ablakra váltana megállna az adatgyűjtés. Erre megoldás lehetne az explicit fókusz kérése (focus grab), de a Wayland esetében erre csak felugró ablak (popup) jellegű, rövid élettartamú ablakok esetében van lehetőség. Továbbá egy ilyen megoldás ellentmondana annak a követelménynek, hogy az alkalmazás a háttérben fusson, grafikus felhasználói felület nélkül.

3.1.2 Miért nem alkalmas a feladatra a Wayland?

A sikertelen próbálkozás után visszatértem a Wayland-el kapcsolatos kutatáshoz és elkezdtem mások által készített alkalmazások forráskódját tanulmányozni. Elsősorban olyan alkalmazásokra koncentráltam, amelyeknél nagy valószínűséggel felmerült az a probléma, amibe én is belefutottam. Többnyire olyan szoftverek forráskódját néztem meg, amelyek képernyő megosztással, remote desktop funkcióval vagy egér / billentyűzet emulálással foglalkoznak. Ezenkívül egy GNOME asztali

környezethez készült widgetet is tanulmányoztam, amely kurzor követéssel foglalkozik (xeyes). A kutatás sajnos kiábrándító eredményeket hozott. Általánosságban három különböző típusú működéssel találkoztam az említett szoftvereknél:

1. Az alkalmazás egyáltalán nem támogatja a Wayland protokollt.
2. Az alkalmazás kísérleti jelleggel, részlegesen támogatja a Wayland protokollt, azaz bizonyos funkciók nem elérhetőek a szoftverben Wayland alatt.
3. Az alkalmazás valamilyen megszorítások mellett támogatja a Wayland-et, például csak GNOME asztali környezeten működik.

Lényegében azt a problémát figyeltem meg, hogy az alap Wayland protokoll meglehetősen karcsú és nem biztosít interfészeket olyan feladatokhoz, mint a képernyőfelvétel vagy a bementi eszköz emuláció. Az ilyen esetekben általában a fejlesztők protokoll kiegészítéseket hoznak létre, amelyet az általuk használt kompozitor implementál és ezután már képes a megfelelő interfészeket nyújtani. Ezért áll fent az a helyzet is például, hogy a csak GNOME asztali környezetet támogató szolgáltatások képesek támogatni a Wayland-et, mert a GNOME által használt Wayland kompozitor implementáció (Mutter) implementál bizonyos protokoll kiegészítéseket, amelyek ezt lehetővé teszik. A helyzet viszont még ennél is bonyolultabb, ugyanis különböző Wayland kompozitor implementációk különböző protokollt kiegészítéseket definiálnak, amelyek egymással általában nem kompatibilisek. Tehát ahhoz, hogy egy olyan alkalmazást készítsek, ami minden Wayland-et használó disztribúción működik vagy az alap protokollt kellene csak használnom, vagy a különböző asztali környezetekhez implementálni az általuk nyújtott interfészeket.

Az alap protokollon belül nincs lehetőség globális kurzor információ lekérdezésére, a különböző protokoll kiegészítések, amelyekkel ezt meg lehetne tenni pedig jelenleg nem állnak még rendelkezésre. Ezért arra jutottam, hogy a Wayland protokoll jelenlegi állapotában nem összeegyeztethető az adatgyűjtő kliens alkalmazás előzetes követelményeivel. A probléma különben nem egyedi, sok alkalmazás esetében megfigyelhető, hogy nehézkes az X-ről való átállás Wayland-re. Én a továbbiakban a kliens alkalmazás implementálása során visszatértem az X Window System protokollhoz.

4 Az alkalmazás architektúrája

Az alkalmazás felépítésének ismertetése, rendszer határok bemutatása, komponensek ismeretetése.

5 Linux kliens alkalmazás

6 Backend szerver

7 Webes vékonykliens

8 Elvégzett munka értékelése

Az elvégzett munka értékelése.

8.1 Tesztelés

Az alkalmazás tesztelésének bemutatása.

9 Összefoglalás

A diplomaterv összefoglalása.

9.1 Konklúzió

A konklúzió ismertetése.

10 Köszönetnyilvánítás

11 Irodalomjegyzék

[1] „Windowing System,” 13 03 2022. [Online]. Available: https://en.wikipedia.org/wiki/Windowing_system. [Hozzáférés dátuma: 03 05 2022].

Függelék