Szűcs Dániel – BSY9BQ

Szeged, 2023.03.31.

|  |  |
| --- | --- |
| 1. | 1 - 40 óra |
| 2019. 03. 26. – 2019. 04. 04. | |
| A szakmai gyakorlatomat az SZTE Szoftverfejlesztési tanszéken végeztem, ahol C++ fejlesztői pozícióban munkálkodtam. Felvételem után az első 40 órában betanuláson vettem részt. A betanulás első részében általános információkkal ismerkedtem meg, mint például, hogyan tudom elérni a belső szervereket, hogyan tudom rögzíteni a munkaóráimat, hol érem el a munkahelyi chatet stb. Ezek után a betanulásra kapott számítógépre kellett Ubuntu-t telepítsek. A betanulás első témája a Linux alapok volt. Itt a GNU szoftvercsomag programjairól tanultam részletesen a BASH-ról, majd a GNU coreutils többi programjáról például: cd, chmod, grep, sudo, make, xargs. Ezen felül tanultam még a Linux kernelről, hogyan tudok magamnak kernelt fordítani, és hogyan lehet kernel modulokat betölteni. A következő részben a különböző függvény könyvtárakról tanultam, mint például a glibc és a libstdc++. Ezen felül tanultam még a virtualizációról is és telepítettem VirtualBox-ba egy Arch Linux disztribúciót próbaként. A második téma betanulásom során a Git verziókezelő rendszer volt. Itt megtanultam hogyan kell bare (remote) repositoryt inicializálni, hogyan kell local repositoryt inicializálni, hogyan tudom a nevemet és e-mail címemet beállítani, hogyan érem a logot stb. Majd a Git komplexebb parancsairól beszéltünk, itt megtanultam hogyan működik a branching, mi az a rebase, merge, cherry-pick, hogyan tudom a változtatásaimat committolni majd feltenni a remote-ba, hogyan tudok conflict-ot feloldani. A harmadik téma a betanulás alatt a GNU toolchain volt. Itt tanultam a gcc fordítóprogramról és különböző kapcsolóiról. Megnéztem, hogy a fordító hogyan rendezi el a programomat memóriában, a különböző változók és a kód milyen bináris szekciókba kerülnek. Megtanultam mi az Address Space Layout Randomization (ASLR). Megtanultam, hogy a toolcahin milyen komponensekből áll, és hogy a komponensek között milyen kapcsolat áll. Ezek után a linker komponenssel foglalkoztam többet, ahol megnéztem hogyan tudok linker scriptet írni, hogyan tudom külső könyvtárak elérési útját megadni neki stb. A betanulás utolsó témája a debugging volt. Itt tanultam külöböző debuggolásra alkalmas programokról, mint például a gdb, cgdb, lldb. Megtanultam hogyan kell breakpointokat és watchpointokat használni, valamint hogyan lehet kivételeket elkapni debuggolás közben. | |
| 2. | 41 - 80 óra |
| 2019. 04. 04. – 2019. 04. 18. | |
| A betanulást követően a munkám egy másik irodában folytattam, ahol megismerkedtem a projekten dolgozó kollégákkal. Első feladatom a számítógépem installálása volt, melyet a hálózat beállítása követett, hogy el tudjam érni VPN-en keresztül a belső hálózatot. Ezután létre kellett hoznom egy új felhasználót, amivel hozzáférek a belső hálózaton elérhető dolgokhoz. Az első programozási feladatom a projekten az volt, hogy statikus analízis segítségével kód hibákat kellett kijavítsak. Ehhez megismertem a Clang-Tidy nevű eszközt, amely a kiemeli a kódban található potenciális problémákat. Mivel a projekt teljes szkennelése körül belül 15 percet vesz igénybe így inkább iteratív módon próbáltam dolgozni, ez viszont csak részben lehetséges a komponensek összefüggése miatt. Ezt a feladatot azért kaptam, hogy a projekt teljes egészét át tudjam tekinteni. A projekten fejlesztett program egy fordítóprogram, ami öt fő komponensből áll, név szerint: Lexer, Parser, Checker, Compiler, Emitter. A következő feladatom során a GitLabon feljegyzett issuekkal kellett foglalkoznom a Parser és a Lexer komponensben. A legtöbb eseteben a hiba az volt, hogy a program rossz sor információt adott vissza, ha valami hibát talált a forráskódban. Ezeket gyorsan tudtam is javítani. Minden issue javítása után szólnom kellett a mentoromnak, aki átnézte a változtatásaimat, majd elmondta a meglátásait. Miután mindent rendben talált feltehettem a kódomat a közös branchre, majd lezártam az issue-t. A következő feladatom az volt, hogy a valósítsam meg a feltételes vezérlési szerkezet lefordítását. Ehhez meg kellett ismerjem a Compiler komponenst. A feltételes vezérlési szerkezet fordítása a következőképpen történik: Először megnézzük, hogy az *if* feltétele fordítási időben kiértékelhető-e, mert ha igen akkor magát a vezérlési szerkezetet le sem kell fordítani ezzel runtime időt és teljesítményt nyerve. Ezek után történik a tényleges fordítás. Először is szükség van egy *label-re* ahová akkor ugrunk, ha az *if* feltétele hamis. Ezután lefordítjuk az *if* feltételét melynek allokálnunk kell egy regisztert, aminek értékét a nullával fogjuk összehasonlítani, majd a végén le kell még fordítani konzekvens utasítást is. Ha van *else* akkor frissíteni kell a *label-t,* hogy a megfelelő helyre történjen a vezérlés átadása. Ha *else if* követi a konzekvens utasítást akkor rekurzívan hívjuk az *if-et* fordító függvényt. A Compilerben vissza kell állítani a következő *label-re* mutató pointert és ezzel gyakorlatilag kész a feltételes vezérlési szerkezet lefordítása. | |
| 3. | 81 - 120 óra |
| 2019. 04. 18. – 2019. 05. 09. | |
| A következő feladatom különböző logikai bináris operátorok lefordítása volt. Ezek az operátorok a következők: ==, !=, >=, <=. Ezeknek az operátoroknak a lefordítása páronként nagyon hasonlóan történik, ezért a kódomat template-ekkel oldottam meg, hogy csökkentsem a kód duplikációt. Mind a négy operátor esetében az első lépés a feltételes vezérlési szerkezethez hasonlóan azzal a lépéssel kezdődik, hogy megnézzük, hogy fordítási idejű konstansok-e, ha igen akkor az előre kiszámolt értéket töltjük be az akkumulátorba. Mivel ezek az operátorok mind logikai értéket adnak ezért ez igaz érték esetén 1, hamis érték esetén 0. A template függvények template paraméterben egy *flag-et* várnak, ami fordítási időben eldönti, hogy melyik operátor kódját szintetizálja. Ezt a munkámat is átnézete a mentorom majd feltöltöttem a közös branchre. Ettől a ponttól kezdve szabadon dolgozhattam, nem kellett a kódomat bemutatni a mentoromnak, hanem a hagyományos code review folyamaton esett át. A következő feladatom a Checker komponenst érintette, ami a projekt legbonyolultabb komponense. A Checker komponens végzi a szemantikai kontextus függő analízist a kódon. Mivel a nyelv, amit a fordítóprogramunk fordít statikusan típusos, és objektum orientált, így nagyon sok különböző szemantikai analízisen esik át. Az én feladatom a *class-okat* érintette, azon belül is az *abstract* függvényekre kellett statikus analízist implementálnom. Az analízis a következő szabályok mentén működik: *abstract* függvénye csak *abstract* osztálynak lehet, *abstract* osztály nem példányosítható, nem *abstract* osztály, ha *abstract* ősből származik akkor köteles minden *abstract* függvényének implementációt adni. Ezek az analízisek relatív könnyen implementálhatók voltak. A következő feladat, amelyet már magamnak választhattam az Emitter komponenst érintette. A fordítóprogram a futása végén nem platform specifikus assembly kódot emittál, hanem bytecode-ot. Ebben a bytecode-ban elhelyezhetők különböző féle metainformációk amelyeket a nyelvhez fejlesztett runtime és debugger fel tudnak használni. A feladat lényege az volt, hogy minden egyes generikus függvényhez, vagy osztályhoz ki kellett emittálni az bytecode-ba, hogy mennyi típus paramétere van, és azoknak van-e egymás között függőségük vagy felső korlátjuk. Ehhez tulajdonképpen minden információ meg van az adott AST node-on, így az Emitter gyakorlatilag csak egy szerializációs feladatot lát el. | |
| 4. | 121 - 160 óra |
| 2019. 05. 09. – 2019. 06. 06. | |
| Még pár bug után elkezdtem foglalkozni komolyabb feladatokkal is. Az első nagyobb JIRA ticketem a projektünk legnagyobb komponensének a refactorolása volt, melyet szét kellett szednem kisebb komponensekre. Erre azért volt szükség, mivel minden alkalommal, amikor arra a komponensre került a fejlesztés legalább 2 órával tovább tartott a fejlesztés, mivel annyira átláthatatlan volt az egész. Azért volt ez jó, mivel nagyban megismertem, hogy hogyan is működik az egész kód és ezt közösen csináltam az egyik munkatásammal, hogy ne kelljen egyedül szenvednem és felelősséget vállalnom, ha véletlenül nem sikerül úgy ahogy. Ez az a komponens, amire a kódunk 80%-a kihat, a view, amit az addin paneljén az adatok megjelenítésére szolgál. A legnagyobb kihívást itt nyílván az jelentette, hogy nem ismertem a kódot és nagyon sok mindent eltörtem, amit nem kellett volna. Szerencsére a tesztelő a csapatunkban mindent rendesen átnézett, így nem kellett nagyon aggódnunk, ha talált valami, akkor megmondta, hogy hol a hiba, nekünk csak azzal kellett foglalkoznunk, hogy megmaradjon az eddigi működés viszont másképpen történjen az elhelyezkedése kódban, ne legyen ennyire olvashatatlan. Egyértelműen nagyon sokszor újra kellett kezdenünk, de végül egy sokkal igényesebb dolog alakult ki belőle. Egy másik feladatom az volt, hogy az általunk előállított gbt kiterjesztésű csatolmányok más fajta névvel legyenek csatolva a levélhez, mint amilyenek voltak. Első ránézésre ez egy egyszerű feladat volt, de a csapatvezetőm kérte, hogy ez is vonjon maga után egy refactorálást, mivel nem volt a csatolmány készítő komponensünk megfelelő állapotban; nem volt eléggé objektumorientált. | |
| 5. | 161 - 200 óra |
| 2019. 06. 06. – 2019. 07. 01. | |
| Ez után a következő feladatom az volt, hogy egy teljesen új funkciót implementáljak az addin-ban. A feladat lényege az volt, hogy e-mail írás során az Outlook jobb oldalon található addin-unkon megjelenő CRM adatok közül, azokra jobb-klikkelve ki lehessen választani egy olyan funkciót, hogy az adott adatrekordot hozzáfűzzük hivatkozásként az éppen írott email-hez. A link nem csak egy egyszerű hivatkozás volt, külön dizájn volt rá. A linkre rákattintva megnyílik a felhasználó böngészőjében a céges oldal, ahol az adott rekord található, így azt jobban meg lehet vizsgálni. Az alapkoncepció az volt, hogy átkonvertáljuk az email törzsét html alapúra, így könnyen be tudjuk rakni magát a UI element-et. Ez azért nem volt jó megoldás, mivel nem tudjuk, hogy az adott felhasználó milyen módon kívánja az email törzsét megadni, emiatt ezt az ötletet el kellett vetnünk. Egy másik ötletünk az volt, hogy generálunk egy képet, amin rajta lesz az adott hivatkozáshoz megjelenítendő cím és majd ezt a képet szúrjuk be és alakítjuk át linkké. Ez a megvalósítás két okból kifojólag sem volt megfelelő. Először azért is, mivel a kép kigenerálása nagyon hosszas ideig tartott és nem várhattuk el a felhasználótól, hogy ennyi ideig várjon.Viszonylag egyszerű fix volt, nem kellett vele sokat foglalkozni. Egyedül az volt nehéz benne, hogy több réteg is volt egymáson, így más színű volt a kijelölés, ezért nem a színekkel kellett foglalkozni egy kicsit. | |
| 6. | 201 - 240 óra |
| 2019. 07. 01. – 2019. 07. 15. | |
| Ezt követően a következő fontos feladatom az volt, hogy egy ún. töltőképernyőt hozzak létre az addin-hoz. Magyarán szólva, addig amíg nem érkeznek meg a szerverről az adatok és az addin még nem áll készen addig ezt a felhasználói felületen jelezzük a felhasználó felé valamilyen formában, például animációval. A feladat megoldása elég hosszúra sikeredett, mivel át kellett strukturálni, formázni az XAML fájlok nagy részét a projektben. Külön nehézséget okozott, hogy az animáció behúzása egy külön bug-ot okozott a felhasználói felületen, ami abban nyílvánult meg, hogy miután véget ért a betöltés, az animált elemek látszólag eltűntek, de ottmaradt a helyükön valami, ami a későbbiekben zavart okozott a hover animációkban. Ezt csak egyféleképpen tudtuk kijavítani: végig kellett iterálni a vizuális fán és kikapcsolni az összes ilyen animált elem animációját. Mivel az addin használata során is szükség van erre a töltőképernyőre, így az nem lett volna ésszerű döntés, hogy minden egyes forrás letöltésekor újra és újra bejárjuk a vizuális fát, ezért egyszerűen nem használtuk az animációt, csak szimplán a szürke alakzatokat hagytuk benne. Ha valaki olyan kezdett volna ezen a feladaton dolgozni, aki már régebb óta dolgozik ezen a projekten és már meg van az a tudása, ami ahhoz kell, hogy ne hasaljon el olyan buktatókon, mint amilyeneken én hasaltam el, akkor egy valószínűleg egyszerű feladat lett volna, amit a megadott időn belül el tudok végezni. | |
| 7. | 241 - 280 óra |
| 2019. 07. 15. – 2019. 07. 22. | |
| A következőkben egy másik általunk fejlesztett addin-ban kellett egy új funkciót leimplementálni. És a mail merge csak annyit csinál, hogy legenerálja az adott mezőnevekkel elkészült dokumentumot annyiszor, ahány ember van és kitölti minden ember megfelelő értékével ezeket a mezőket. Ez például hasznos tud lenni, ha sok embernek akarunk feladni egy levelet ugyanazzal a mintával a postán. Az addin-unk is valami hasonlót csinál, de az csak az általunk készített adathalmazt dolgozza fel. A feladatom az volt, hogy leimplementáljak egy olyan funkciót, amely elérhetővé teszi azt a választási lehetőséget, hogy melyik mező alapján rakja a készült dokumentumok sorrendbe. A legnagyobb kihívást az okozta, hogy hol találjam azt a komponenst, ami a csv adatok betöltéséért felelős, illetve hova rakjam azt a logikát, ami azt dönti el, hogy milyen sorrendbe legyenek megcsinálva a merge-k. A static, ill. dynamic mail merge különböző helyeken volt a kódban. Először nem is tudtam, hogy létezik másfajta mail merge. Emiatt a tesztelés után visszakerült hozzám, hogy megcsináljam mindkettőre egységesen. | |
| 8. | 281 - 320 óra |
| 2019. 07. 22. – 2019. 07. 26. | |
| Végezetül, mivel egyszerre több projektet is magában foglal a solution, amit fejlesztünk, így célszerű volt a közös részeket összevonni. Munkatársam leimplementálta azt a logikát, amivel sokkal egyszerűbben el tudjuk dönteni, hogy melyik terméket használjuk éppen. Az én feladatom az volt, hogy azokat az osztályokat, amiknek már nem volt értelme már, hogy külön-külön osztályokban szerepeljenek, összevonjam egy közös osztályba a közös projectbe így mindegyik alkalmazás ugyanazt az egységes logikát használja. Nem volt nagy eltérés a kódok között, mivel nagyjából ugyanazt csinálták, így meglehetősen jól lehetett vele haladni Elég egyszerű feladat volt, viszont nagyon sok tesztjavítási ellátnivalót vont maga után. Valamint tesztelőnek is nagyon sok idő volt mire mindent megnézett, hogy ugyanúgy működik-e minden, mint ahogyan eddig. Ezek után kijavítottam még elég sok support ticketet. A support ticketek a felhasználóktól jönnek a német kollégáinknak és ha azt hibának vélik, akkor továbbítják nekünk visszajelzésként. Arról számolnak be, hogy éppen melyik funkcióban van egy kis hiba. Általában egy részletes log fájllal csatolva érkeznek és a hiba elég pontosan meg van fogalmazva. A feladatot megnehezítette, hogy nem ismertem még az eventek működését és sokat kellett kutatnom, hogy ezt éles alkalmazásokban hogyan szokták megcsinálni. A megoldásra más, kifejezetten erre célra elkészített eventeket használtam és egy olyan WPF konstrukciót, amellyel event-eket lehet küldeni command-ként a ViewModel-ekre, így valóban megvalósul az MVVM tervezési minta és a funkció is működőképes lesz. | |