VILNIAUS UNIVERSITETAS

MATEMATIKOS IR INFORMATIKOS FAKULTETAS

PROGRAMŲ SISTEMŲ KATEDRA

Modelinės operacinės sistemos kūrimas kompiuterio architektūros abstrahavimo metodu

Model operating system development using computer architecture abstraction method

Bakalauro darbas

Atliko: Povilas Panavas (parašas)

Aurelijus Rožėnas (parašas)

Darbo vadovas: doc. dr. Antanas Mitašiūnas (parašas)

Vilnius – 2009

Santrauka

Darbo tikslas yra ištirti operacinių sistemų dalyko praktinių užsiėmimų organizavimo metodikas ir pasiūlyti bei parengti modelinių operacinių sistemų realiai architektūrai sukūrimo metodiką ir ją remiančią medžiagą, minimizuojančią techninio darbo sąnaudas.

Darbas susideda iš dviejų pagrindinių dalių. Pirmoje darbo dalyje atlikta dėstymo metodikų analizė, apibrėžti vertinimo kriterijai, pateikti įvertinimai. Atsižvelgiant į gautus rezultatus, pateiktos dvi alternatyvos dabar naudojamoms metodikoms. Tuomet jos palyginamos su kitomis ir išrenkama geriausia alternatyva.

Antroje darbo dalyje pateikiama pasirinktą metodiką remianti medžiaga. Ši dalis susideda iš kūrimo aplinkos ir tinkamų įrankių aprašymo bei žingsninio vadovo, leidžiančio susipažinti su Intel x86 architektūros veikimu. Parengta metodika ir remianti medžiaga leidžia greitai įsisavinti aparatūros programavimo principus, tokiu būdu minimizuojant studentų techninio darbo sąnaudas.

Raktiniai žodžiai: operacinių sistemų dalykas, reali architektūra, techninio darbo minimizacija, operacinių sistemų kūrimo metodikos, operacinių sistemų kūrimo įrankiai, aparatūrinio lygio abstrahavimas.

Summary

The goal of this work is to explore practical training methods of the subject of operating systems and to propose and create a methodology of model operating systems development for real architecture and also to provide supporting material, which minimizes costs of technical work.

The work consists of two main parts. The first part consists of analysis of teaching methodologies, defining of evaluation criteria, and assessment. Given the results obtained, the two alternatives are proposed for currently used methodologies. Then all the methodologies are compared and the best alternative is selected.

The second section provides material for supporting the chosen methodology. This part is composed of the development environment and descriptions of appropriate tools and also manual, enabling students to familiarize with operation of the Intel x86 architecture. This new methodology and supporting material allows to quickly absorb the principles of hardware programming, thus minimizing the cost of technical work for students.

Keywords: subject of operating systems, real architecture, minimization of technical work, methodologies of operating systems development, tools of operating systems development, architecture abstraction.

Turinys

1 Įvadas

Praktiškai visuose universitetuose yra dėstomas operacinių sistemų dalykas. Juose, siekiant įtvirtinti studentų žinias, į pagalbą pasitelkiamos praktinės užduotys. Metodikų organizuoti tokį mokymą yra daug, tad bėgant metams mokymo metodikos tobulėjo bei jų įvairovė augo. Galima išskirti šias pagrindines operacinių sistemų praktinio užsiėmimo metodikų grupes: sistemos kūrimas, pasitelkiant aparatūrinės įrangos simuliatorius, operacinės sistemos išeities tekstų analizė ir modifikavimas bei operacinės sistemos kūrimas realiai architektūrai.

Operacinės sistemos kūrimas realiai architektūrai per daug apsunkina praktines užduotis. Kita vertus, atsiribojimas palieka spragas studentų žiniose. Todėl pasirenkamas tarpinis variantas, kuomet prisirišama prie konkrečios architektūros, tačiau ji yra supaprastinta bei pritaikyta mokymo tikslams [MD99]. Taip pat galimas tik programinis architektūros simuliavimas, kuris naudojamas Mičigano universitete [Gon94].

Kita šios metodikų grupės atmaina yra naudojama Maristo koledže [Sha07]. Praktinių užsiėmimų metu patys studentai simuliuoja kompiuterio architektūrą ir kuria jai dedikuotą sistemą. Vienas svarbiausių bruožų yra tinkamas modelinės operacinės sistemos kūrimo skaidymas į atskiras ir nepriklausomas dalis.

Šiuo metu yra sukurta keletas, mokymo tikslams skirtų operacinių sistemų, pavyzdžiui, „Minix“ ar „NachOs“. Jose nesunkiai galima vykdyti dažniausiai sistemose atliekamus darbus. Tačiau tai yra didžiausias jų trūkumas, nes kuo daugiau galimybių, tuo sunkiau suprasti ir išgryninti operacinės sistemos veikimo principus. Šio darbo tikslas yra pateikti, kiek galima paprastesnę multiprograminę operacinę sistemą. Tokiu būdu dėmesys sutelkiamas į esmę, o ne pagalbinius dalykus.

„PortOs“ ar kitos egzistuojančios sistemos naudojimas operacinių sistemų kursų metu patenka į operacinių sistemų analizės ir modifikavimo metodikų grupę naudojamą Kornelio universitete [AS02, CF88, DSX05]. Sunku realizuoti visą sistemą, todėl pasirenkamas trumpesnis kelias, kuomet praktinių užduočių metu studentai turi keisti vieną ar daugiau sistemos veikimo aspektų.

Sistemos kūrimas egzistuojančiai architektūrai naudojamas Jeruzalės universitete [Frei08]. Metodo ypatumai: maksimalus veikimo principų aprėpimas bei ilga trukmė, atsirandanti dėl didelio programavimo darbų sudėtingumo. Šiame darbe pateikiamas metodas taip pat priklauso paskutiniajai grupei.

Autorių kursiniame darbe [PR08] buvo pateikta modelinė sistema, kuri emuliuoja kompiuterio architektūrą. Tai leido supaprastinti sistemos kūrimą ir veikimo principų

išgryninimą. Šiame darbe žengiamas žingsnis į priekį ir priartėjama prie realios architektūros. Toks metodas leidžia aprėpti maksimalų operacinių sistemų veikimo aspektų spektrą, tačiau turi vieną esminį trūkumą – studentams būna sunku per duotą laiką realizuoti sistemą.

Dažniausiai paskaitų metu būna pateikiama daug teorinės medžiagos, tačiau studentams sunkiai sekasi ją pritaikyti savo laboratoriniuose darbuose.

Darbo tikslas yra ištirti, įvertinti ir palyginti universitetuose esamas operacinių sistemų dalyko praktinių užsiėmimų vykdymo metodikas bei pasiūlyti ir parengti naują metodiką ir jos įgyvendinimą remiančią medžiagą.

Darbui keliami šie uždaviniai:

1. atlikti universitetinių informatikos krypties operacinių sistemų praktinių užduočių organizavimo metodikų sisteminimą;
2. pasiūlyti naujas alternatyvas egzistuojančioms praktinių užsiėmimų organizavimo metodikoms;
3. apibrėžti metodikų vertinimo kriterijus;
4. įvertinti metodikas;
5. pateikti pasirinktą alternatyvą remiančią medžiagą.

Pirmoji darbo dalis apima metodikų analizę ir vertinimą, remiantis darbe apibrėžtais kriterijais. Taip pat šioje dalyje pateiktos siūlomos alternatyvos, kurios palygintos su kitomis metodikomis ir išrinkta geriausia alternatyva.

Antroji darbo dalis žingsninis vadovas, kaip programuoti sistemą Intel x86 kompiuterio architektūrai. Aptariami visi klausimai, į kuriuos reikia turėti atsakymą, norint sukurti sistemą. Pradedant, patogaus būdo užkrauti kuriamą sistemą radimu ir metodų testuoti pasirinkimu iki pertraukimų lentelių perrašymo. Todėl reikalingas laikas sistemos kūrimui žymiai sutrumpėja.

2 Operacinių sistemų praktinių užsiėmimų dėstymo metodikos

Operacinių sistemų kursas yra plačiai naudojama priemonė visame pasaulyje studentų žinioms kompiuterių moksluose gilinti, todėl yra išbandyta daugybė būdų, siekiant kuo didesnio žinių įsisavinimo ir sistemos veikimo principų suvokimo.

Šiame skyriuje apibrėžiami vertinami metodikų aspektai ir jų kriterijai. Vėliau jais remiantis lyginami operacinių sistemų praktinių užsiėmimų mokymo būdai. Taip pat išrenkami visoms metodikoms bendri principai, kurie didina kurso sėkmingumą. Po to aprašomas darbe siūlomas metodas bei jo pranašumai.

2.1 Vertinami aspektai

Darbe dėmesys sutelkiamas į praktinių užduočių atlikimo užsiėmimų metu efektyvumą. Svarbu, kad studentai spėtų per paskirtą laiką atlikti užduotis, bei, kad kuo daugiau teorinių žinių būtų pritaikyta praktiškai.

Paprastai operacinių sistemų kursas turi standartinį kiekį kreditų. Tai reiškia, kad vienas studentas užduotims turi sugaišti ne daugiau nei mėnesį (1 kreditas apytiksliai yra 40 valandų). Studentų grupės dydis neturi būti didesnis už 3 asmenis. Operacinė sistemos atskiri komponentai tampriai susiję, todėl didesnis studentų grupėse skaičius neleistų padidinti atliekamo darbo kiekio. Tą patį rodo ir visos egzistuojančios metodikos, nė vienoje iš jų studentų grupė nesusideda iš daugiau nei 3 studentų.

2.2 Vertinimo kriterijai

Siekiant objektyvaus įvertinimo buvo įvesti šie vertinimo kriterijai:

- praktinių užduočių atlikimo trukmė;
- operacinių sistemų veikimo aspektų kiekis, kurį studentai pritaiko praktinių užduočių metu;
- kaina, reikalinga užsiėmimų organizavimui.

Trukmė matuojama vieno studento darbo mėnesiais. Vienas studento darbo mėnuo lygis vienam balui.

Sistemų veikimo aspektai vertinami procentais. Viso jų yra 32, tad įvertis 50%, reiškia, kad 16 aspektų iš 32 buvo įgyvendinti atliekant praktines užduotis. Procentinį įvertį padalinus iš dešimties gaunamas įvertinimas balais. Taigi, jei metodika apima 53% aspektų, jos įvertis yra 5,3 balo.

Kaina vertinama balais nuo 1 iki 4, kur 1 reiškia, kad nereikia papildomų investicijų ir laiko, 2 – reikia daugiau nei 30 minučių darbo vietos pritaikymui, 3 – reikia už studento darbo vietą mokėti daugiau nei 50 litų, 4 – antras ir trečias įverčiai kartu, t.y., reikia daug papildomo laiko ir pinigų.

Norint susieti šiuos dydžius ir gauti bendrą įvertį taikoma formulė: A + 2 – T – K. Čia A yra aspektų įvertis balais, T – trukmės įvertis balais, K – kaina balais nuo 1 iki 4.

Tokia formulė pasirinkta todėl, kad tobula mokymo metodika apimtų 100% aspektų (10 balų), truktų 1 mėnesį (1 balas) ir kainos įvertis būtų 1 (1 balas). Tokios mokymo metodikos įvertis būtų: 10 + 2 – 1 – 1 = 10.

Kaip jau minėta ankstesniame skyrelyje, svarbu, kad sistemas kuriančios komandos nebūtų didesnės kaip trys asmenys. Didesnis narių skaičius neatneša spartesnio rezultato, nes operacinė sistema yra vienalytis darinys, kurio atskirus komponentus programuoti, prieš tai nesuprogramavus ankstesnių, yra sudėtinga. Atsižvelgiant į šį reikalavimą, buvo įvesta nuobauda – jei trukmė ilgesnė nei 3 mėnesiai, tuomet iš galutinio įvertinimo balais reikia atimti vienetą.

2.3 Metodikų grupės

Visus mokymo būdus galima sugrupuoti į tris kategorijas:

- sistemos kūrimas, naudojimasis aparatūrinės įrangos simuliatoriais;
- egzistuojančios sistemos analizė ir modifikavimas;
- sistemos kūrimas konkrečiai platformai.

Kituose skyriuose kiekviena iš metodikų apibūdinta detaliai, pateikiant jos trūkumus, privalumus ir įvertinimą, naudojantis skyrelyje „Vertinimo kriterijai“ įvestais kriterijais.

2.4 Mokymo metodų evoliucija

Pirmasis plačiai paplitęs dėstymo būdas buvo visiškai teorinis. Studentai būdavo supažindinami su pagrindinėmis sąvokomis, esminiais operacinių elementais, pavyzdžiui, atminties valdymo metodai, įvesties/išvesties įrenginiai, multiprogramiškumas ir panašiai. Pastebėjus, kad studentams sunkiai sekasi įsisavinti žinias, buvo pradėta analizuoti konkrečią egzistuojančią operacinę sistemą. Buvo bandoma pasirinkti, kuo paprastesnę. Kita vertus, galima teigti, kad tuometinės operacinės sistemos būdavo paprastesnės, nes turėdavo, kur kas mažesnį funkcionalumą. Vienas esminių šio metodo trūkumų buvo tai, kad realios sistemos beveik visada veikia ne pagal teorinį, universitetuose dėstomą modelį.

Šiam metodui evoliucionuojant, 1987 buvo išleista knyga „Operating systems design and implementation“. Joje buvo rašoma apie „Minix“ sistemą, kuri buvo specialiai sukurta mokymo tikslams, todėl jos išeities tekstai tapo puikia mokymo priemone. Šis knygos ir operacinės

sistemos derinys buvo toks sėkmingas, kad visiškai pašalino tik teorinį kursų dėstymo modelį. Verta paminėti, kad ši sistema buvo ir yra nuolat tobulinama. 2006 metais pasirodė trečiasis šios knygos leidimas, aprašantis „Minix 3“ veikimo principus.

Sekantis žingsnis buvo perėjimas nuo konkrečių sistemų nagrinėjimo ir modifikavimo prie jų kūrimo. Būtina pažymėti, kad kurti operacinę sistemą, trukdavo ilgai bei dažnai nepavykdavo, todėl netrukus buvo nuspręsta, naudoti supaprastintą aparatūrinę įrangą arba programiniu būdu simuliuoti jos veikimą.

2.4.1 Sistemos kūrimas, naudojantis aparatūrinės įrangos simuliatoriais

Šio metodo esmė, sukurti operacinę sistemą, kuri veikia pateiktoje architektūroje. Egzistuoja trys šio metodo variacijos: specialiai mokymo kursams sukurta supaprastina aparatūrinė įranga [MD99], programinis kompiuterio architektūros simuliavimas [Gon94] bei simuliatoriaus ir jam skirtos sistemos kūrimas [Sha07].

2.4.1.1 Supaprastinta kompiuterinė architektūra

2.4.1.1.1 Privalumai

- reikalinga mažesnė trukmė sistemai dėl supaprastinimo;
- naudojant visiems studentams bendrą architektūrą, paliekama mažiau vietos jų klaidoms, nei leidžiant, kiekvienam iš jų susikurti programinį simuliatorių;
- lengviau ir greičiau išspręsti kai kurias iškilusias problemas, nes visos studentų grupės dirba su identiška platforma.

2.4.1.1.2 Trūkumai

- sukurti bei pagaminti tokią sistemą yra sudėtinga ir brangu;
- studentams sunku gauti priėjimą testavimui prie sistemos bet kuriuo metu.

2.4.1.1.3 Kriterijų įverčiai

- trukmė – 3 mėn. (3 balai);
- aspektai – 80% (8 balai);
- kaina – 4 (4 balai);
- bendras įvertis – 3 balai.

2.4.1.2 Programiniai aparatūrinės įrangos simuliatoriai

2.4.1.2.1 Privalumai

- galima teigti, kad nemokamas metodas;

- supaprastina operacinės sistemos kūrimą, nes atsisakoma sudėtingų šiuolaikinių architektūrinių aspektų;
- galimybė naudojant virtualias mašinas, studentams dirbti su keliomis skirtingomis platformomis;
- lengvesnis kompiuterių administravimas (studentams nereikia administratoriaus teisių).

2.4.1.2.2 Trūkumai

- dirbama supaprastintomis sąlygomis, todėl lieka neaprėptų sistemos veikimo aspektų;
- nėra paplitusių tokio tipo simuliatorių;
- studentams prastai sekasi integruoti savo sistemas su architektūros simuliatoriumi.

2.4.1.2.3 Kriterijų įverčiai

- trukmė – 2 mėn. (2 balai);
- aspektai – 70% (7 balai);
- kaina – 2 (2 balai);
- bendras įvertis – 5 balai.

2.4.1.3 Savarankiškai simuliuojant kompiuterio architektūros veikimą

2.4.1.3.1 Privalumai

- leidžia studentams savarankiškai susikurti norimą architektūrą, todėl pagerėja sistemos darbo principų suvokimas;
- paprasta realizacija. Studentas pats simuliuoja architektūrą ir kuria operacinę sistemą, todėl architektūros ir sistemos sąveika nesukelia problemų;
- nemokamas metodas.

2.4.1.3.2 Trūkumai

- architektūros apibrėžimas dažnai būna atliekamas mechaniškai, t.y. paima bet kuri po ranka pasitaikiusi architektūra ir realizuojamas jos simuliatorius;
- ilgesnė pradinio etapo trukmė, nes studentams reikia ne tik sukurti operacinę sistemą bet prieš tai apsibrėžti architektūrą ir sukurti jos simuliatorių.

2.4.1.3.3 Kriterijų įverčiai

- trukmė – 2 mėn. (2 balai);
- aspektai – 75% (7,5 balo);
- kaina – 2 (2 balai);

- bendras įvertis – 5,5 balai.

2.4.2 Egzistuojančios sistemos analizė ir modifikavimas

Praktinių užduočių metu studentams liepiama modifikuoti vieną ar daugiau sistemos dalių, pavyzdžiui, atminties valdymą [AS02, CF88, DSX05].

Vienas svarbiausių šio metodo aspektų yra pasirinkti tinkamą operacinę sistemą. Tam puikiai tinka specialiai mokymo tikslams sukurtos operacinės sistemos, pavyzdžiui, „NachOs“ ar „Minix“.

2.4.2.1 Privalumai

- Pasirinktos operacinės sistemos dažniausiai veikia ant realios architektūros, tai teoriškai leidžia aprėpti visus kurso aspektus. Tačiau praktiškai būna pasirenkama viena ar dvi sistemos dalys, į kurias įsigilinama.
- Yra keletas operacinių sistemų, kurių veikimas detaliai aprašytas, todėl studentai lengvai savarankiškai gali užkaišyti žinių spragas.
- Kur kas paprastesnis ir mažiau trunkantis metodas nei naujos operacinės sistemos sukūrimas pasirinktai architektūrai.

2.4.2.2 Trūkumai

- Studentai gerai įsisavina tik tai, ką turėjo modifikuoti praktinių užsiėmimų metu. Taigi, didžioji sistemos veikimo principų dalis lieka neaprėpta.
- Sunku surasti tinkamą operacinę sistemą, kurią būtų galima naudoti pratybų metu. Dažna tokia sistema, jau geba atlikti daug įvairių užduočių, šios galimybės paslepia esminius veikimo principus. Kita vertus, sudėtinga rasti tinkamą paprastą sistemą, nes jos būna prastai dokumentuotos.
- Lieka įsisavintas konkretus metodas, o ne jų visuma. Pavyzdžiui, operacinė sistema gali pasirinkti įvarius atminties valdymo metodus, tad kyla klausimas, ar studentas mokės vieną konkretų, ar suvoks, kokios problemos turi būti įveiktos, norint sukurti vienokį ar kitokį metodą.

2.4.2.3 Kriterijų įverčiai

- trukmė – 1 mėn. (1 balas);
- aspektai – 30% (3 balai);
- kaina – 1 (1 balas);
- bendras įvertis – 3 balai.

2.4.3 Sistemos kūrimas egzistuojančiai architektūrai

Šis metodas neapima sistemos kūrimo, jau egzistuojančiam programiniam aparatūrinės įrangos simuliatoriui. Apie tokį metodą skaitykite skyriuje „2.4.1.2 Programiniai aparatūrinės įrangos simuliatoriai“.

Tai metodas, kuomet studentai prisiriša prie konkrečios architektūros, t.y., kuria visiškai programiškai savarankišką sistemą, veikiančia pasirinktoje aparatūrinėje įrangoje [Frei08]. Taip pat nesinaudoja jokiais specialiai tam skirtais įrankiais, pavyzdžiui, bibliotekomis. Tokios sistemos sukūrimas reikalauja daugiausia sąnaudų lyginant su kitais metodais.

2.4.3.1 Kūrimas egzistuojančiai architektūrai be pagalbinių priemonių

2.4.3.1.1 Privalumai

- visų operacinių sistemų veikimo principų aprėpimas;
- didesnis pasitenkinimas sukūrus tokią sistemą.

2.4.3.1.2 Trūkumai

- Sudėtinga sukurti. Norint pagaminti savo sistemą, neužtenka tik teorinių žinių apie veikimo principus. Reikia žinoti, kaip patogiai užkrauti sistemą, kaip ją testuoti ir dar daugybę papildomų dalykų.
- Praktiškai negalima panaudoti aukšto lygio programavimo kalbų, tokių kaip Java ar C#. Todėl kūrimo procesas tampa dar sudėtingesnis ir lėtesnis.
- Ilga trukmė.

2.4.3.1.3 Kriterijų įverčiai

- trukmė – 6 mėn. (6 balai);
- aspektai – 100% (10 balų);
- kaina – 1 (1 balas);
- bendras įvertis – 6 balai.

2.5 Bendri principai

Pasirinkus bet kurį metodą, labai svarbu, kiek atskirų užduočių pateikiama studentams. Svarbiausia, kad tos užduotys būtų kuo mažiau priklausomos viena nuo kitos. Nustatyta, kad į kuo daugiau atskirų fazių suskirstomos užduotys, tuo studentai pasiekia geresnių rezultatų.

Dažnai būna sunku užduotis atskirti vieną nuo kitos. Pavyzdžiui, šiuo metu aktualios tik multiprogramines operacinės sistemos, todėl paskutinė užduotis galėtų būti multiprogramiškumo įgyvendinimas. Tačiau norint tai pasiekti, reikia realizuoti tiek virtualią atmintį, tiek kanalų

operacijas, tiek patį multiprogramiškumą. Daryti kurį nors iš šių komponentų atskirai neparanku, nes vienas be kito jie neduoda nieko naujo kuriamai sistemai.

Kita dažnai sutinkama rekomendacija – programavimo darbus pradėti kuo anksčiau. Galima pirmajai užduočiai skirti ką nors ne tiesiogiai susijusios su pagrindinėmis kurso užduotimis. Tokia užduotis turėtų būti tarsi apšilimas, leidžiantis vėliau be didesnių problemų imtis kitų praktinių uždavinių sprendimo.

Visus šiuos patarimus galima aprėpti vienu sakiniu: studentų užduotims turi būti uždėti aiškiai apibrėžti rėmai. Tokiu atveju studentai vienu metu dirba tik su vienu operacinių sistemų aspektu, todėl pasiekia kur kas geresnių mokymosi rezultatų.

2.6 Darbe siūlomos praktinių užsiėmimų metodikų alternatyvos

Pagrindinis praktinių užsiėmimų tikslas įtvirtinti teorines žinias atliekant paskirtas užduotis. Visiškai šią užduotį įvykdo tik viena metodiką – naujos sistemos kūrimas egzistuojančiai architektūrai. Ši metodika yra darbe siūlomų alternatyvų pagrindas. Kadangi vienintelis jos trūkumas per ilga trukmė, darbe siūlomi būdai orientuoti į reikalingo laiko kiekio sumažinimą.

Lėtą studentų operacinių sistemų projektų įgyvendinimą lemia sudėtingumas, kylantis iš didelio aprėpiamų aspektų skaičiaus. Reikia ne tik suprogramuoti sistemą, bet išmanyti papildomus dalykus, pavyzdžiui, kaip patogiai užkrauti ar testuoti sistemą.

Abi siūlomos alternatyvos sumažina reikalingą trukmę, pateikdamos dalinius ar visiškus sprendimus tam tikriems aspektams, pavyzdžiui, darbui reikalingi programiniai įrankiai, testavimo būdai, patogi programavimo terpė, metodas duomenims iš klaviatūros skaityti, metodas duomenims į ekraną išvesti ir panašiai.

2.6.1 Pirmoji alternatyva – operacinės sistemos kūrimo surinkimo būdu metodika

Darbe aprašyti žingsnius nuo naudojamų įrankių iki galimybių realizuoti kažkokį konkretų sistemos komponentą vienokiu ar kitokiu būdu. Norint tai pasiekti, darbe pateikti susistemintą informaciją, leidžiančią be didesnių problemų pradėti savo operacinės sistemos kūrimą. Pagrindiniai tikslai yra sutrumpinti reikalingą laiką sistemos realizacijai ir maksimaliai išplėsti veikimo principų aspektų aprėpimą kurso metu.

Metodui paremti sukurti multiprograminę operacinę sistemą., kuriai keliami šie reikalavimai:

- paprastumas - turi būti lengva išgryninti sistemos veikimo principus;
- išeities tekstai turi būti gerai dokumentuoti ir lengvai skaitomi;

- kiekvienas sistemos aspektas turi būti detaliai aprašytas;
- naudojantis šiuo darbu ir išeities tekstais, turi būti lengva sukurti naują sistemą.

Įgyvendinus aukščiau aprašytus reikalavimus, studentams reikalinga projekto atlikimo trukmė sumažėtų iki trijų procentų, o aprėpiamų aspektų skaičius teoriškai išaugtų iki 100%. Taip įvyktų todėl, kad darbe būtų pateikti sprendimai nuo užkrovimo ir testavimo būdų iki taimerio sukūrimo. Taigi, studentai turėtų teorines žinias paremiantį pavyzdį bei įrankius ir būdus naujai sistemai sukurti.

Atidžiau panagrinėjus šį metodą, galima pastebėti, kad jis labai panašus į egzistuojančios sistemos analizę ir modifikavimą. Panašumas lemia tai, kad nors teorines aprėpiamų aspektų skaičius turėtų būti 100%, tačiau praktiškai jis būtų 20-100%, priklausomai nuo studentų sąžiningumo. Bet kuris studentas galėtų paimti veikiantį pavyzdį ir jį modifikuoti vietoje to, kad remdamasis čia pateikta medžiaga sukurtų savo sistemą. Aptikus šį trūkumą buvo nuspręsta ieškoti kitos alternatyvos.

2.6.1.1 Įvertinimas

- trukmė – 3 mėn. (3 balai);
- aspektai – 60% (6 balai);
- kaina – 1 (1 balas);
- bendras įvertis – 4 balai.

2.6.2 Antroji alternatyva – operacinės sistemos kūrimas kompiuterio architektūros abstrahavimo metodas

Šis būdas panašus į pirmąją alternatyvą, tačiau vietoje veikiančios sistemos pavyzdžio, pateikiama parodomoji programa, kurioje nepilnai įgyvendinti dauguma sistemos veikimui reikalingų dalių. Pavyzdžiui, programa paspaudus klaviatūros klavišą, jo simbolį išveda į ekraną. Taigi, studentas turi teorinę medžiagą, kaip veikia klaviatūra bei veikiančią programą, kuri pagrindžia teorinę medžiagą. Tačiau svarbiausias metodikos ypatumas yra tas, kad tokiu būdu studentui pateikiami tik įrankiai, kuriais naudojantis galima sukurti operacinę sistemą egzistuojančiai kompiuterio architektūrai. Pavyzdžio atveju tai reiškia, kad jis turi programos dalį, kuri geba skaityti duomenis, tačiau pats privalo išplėsti ją iki to, kad duomenys būtų skaitomi prasmingomis dalimis ir perduodami sistemoje vykdomoms programoms.

Taip pat darbe aprašyti, kaip teoriškai papildyti programą iki pilnai funkcionuojančios operacinės sistemos. Tai reiškia, kad prie pateikiamų įrankių pridedama teorinė medžiaga, kaip juos išplėtoti bei papildyti iki išbaigtos sistemos.

Metodui paremti sukurti parodomąja programą, kuriai keliami šie reikalavimai:

- Dalinis operacinės sistemos komponentų realizavimas. Pateikiamas ne pilnas sprendimas, o tik ta dalis, kuri susijusi su šio metodo sudėtingumu, kylančiu dėl tiesioginio darbo su aparatūrine įranga. Jei pasitelktume anksčiau pateiktą pavyzdį, tuomet tai reiškia, kad pateikiamas būdas skaityti iš aparatūros įvedamus duomenis (klaviatūra), tačiau tuos duomenis įprasminti, pateikdami vykdomoms sistemos programoms ar kaip komandą pačiai sistemai, privalo studentai.
- Išeities tekstai turi būti gerai dokumentuoti ir lengvai skaitomi.
- Kiekvienas sistemos aspektas turi būti detaliai aprašytas.

Įgyvendinus aukščiau aprašytus reikalavimus, studentams reikalinga projekto atlikimo trukmė sumažėtų iki trijų mėnesių, o aprėpiamų aspektų skaičius teoriškai išaugtų iki 100%. Taip įvyktų todėl, kad darbe būtų pateikti sprendimai nuo užkrovimo ir testavimo būdų iki taimerio sukūrimo. Taigi, studentai turėtų teorines žinias paremiantį pavyzdį bei įrankius ir būdus naujai sistemai sukurti.

2.6.2.1 Įvertinimas

- trukmė – 3 mėn. (3 balai);
- aspektai – 100% (10 balų);
- kaina – 1 (1 balas);
- bendras įvertis – 8 balai.

2.6.3 Siūlomų metodų palyginimas

Abi alternatyvos turi daug bendra. Tiek viena, tiek kita, pateikia įrankius bei būdus nuo A iki Z, t. y., tiek įrankius bei būdus sistemos programavimui ir testavimui, tiek teorines žinias bei jas grindžiančius pavyzdžius.

Pažvelgus į abiejų alternatyvų bendrus įvertinimus skirtumas akivaizdus. Antroji lenkia pirmąją dvigubai. Tai lemia, jog nors pirmojo metodo teorinis aprėpiamų sistemos ypatumų kiekis turėjo būti maksimalus, tačiau praktiškai tėra 60%. Tuo tarpu antrojoje tiek teorinis, tiek praktinis aprėpiamų aspektų skaičius yra maksimalus.

Turint aukščiau minimus siūlomų alternatyvų ypatumus, buvo nuspręsta pasirinkti antrąjį metodą bei darbe pateikti jį remiančią medžiagą.

2.7 Visų metodikų palyginimas

1 lentelėje surašyti visi vertinimo kriterijų įverčiai kiekvienai darbe minimai metodikai. 2 pav. pateiktas grafinis palyginimas, kuriame dėl patogaus grafinio pateikimo vienas iš kriterijų –

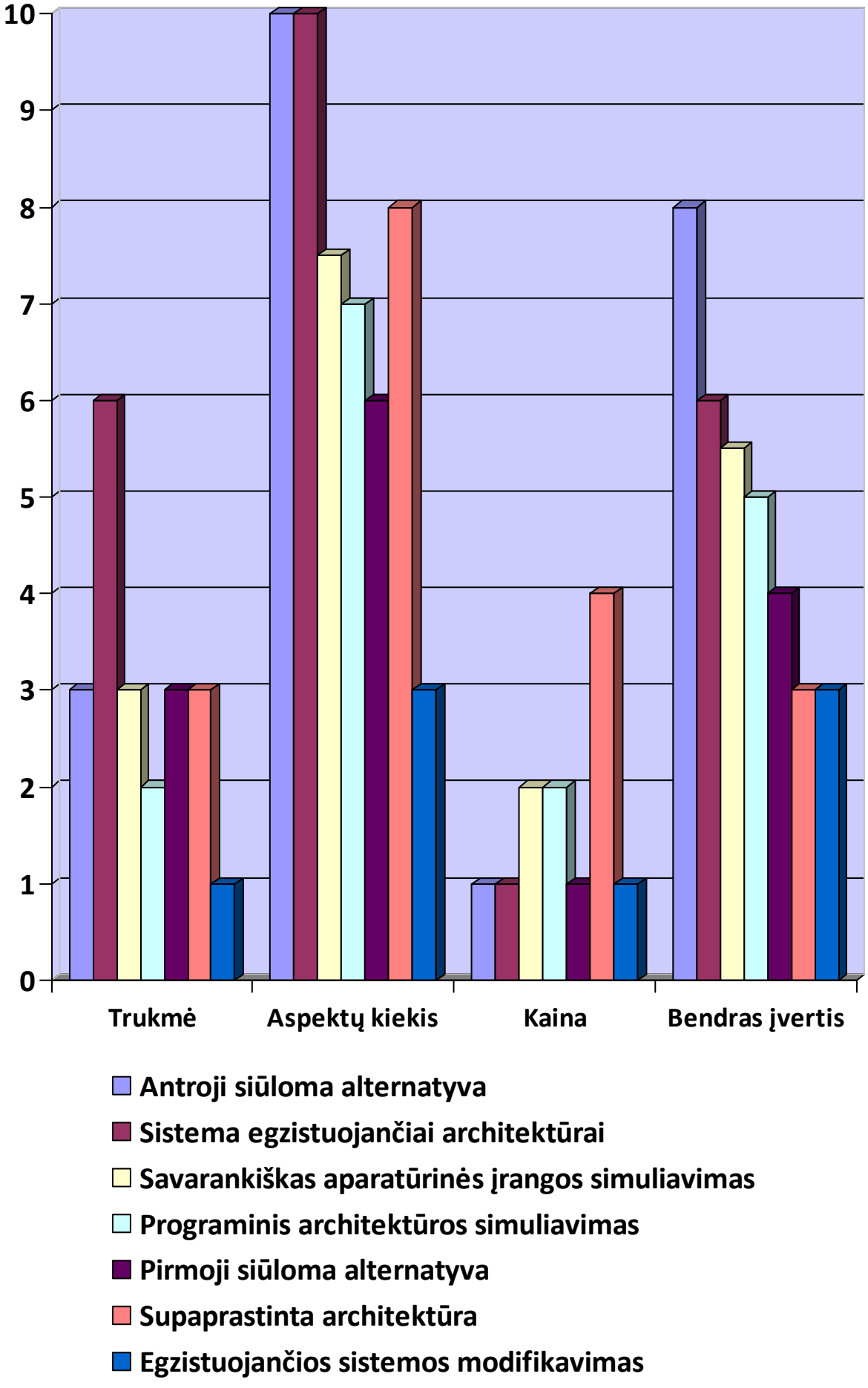
aspektų skaičius – atvaizduotas į balus nuo 1 iki 10. Kur balų skaičius reiškia procentinę jo vertę padalintą iš dešimties.

Dėl metodikoms keliamų reikalavimų, kuo daugiau aprėpiamų aspektų per trumpesnį laiką ir už mažesnę kainą, aukščiausią bendrą rezultatą turi antroji siūloma alternatyva. Šalia jos, antroje vietoje, yra abiejų alternatyvų pagrindą sudaranti metodika – sistemos kūrimas egzistuojančiai architektūrai.

1 lentelės pabaigoje, kaip ir galima buvo tikėtis, atsižvelgiant į keliamus reikalavimus, yra pirmoji siūloma alternatyva ir egzistuojančios sistemos modifikavimas, tai lemia per mažas aprėpiamų sistemos veikimo ypatumų skaičius. Tarp šių autsaiderių patenka supaprastintos architektūros naudojimas, kurio prastą rezultatą nulemia didelė kaina.

1 lentelė – mokymo metodikų įverčiai

Metodika	Trukmė balais	Aspektų kiekis balais	Kaina balais	Bendras įvertis balais
Antroji siūloma alternatyva	3	10	1	8
Sistema egzistuojančiai architektūrai	6	10	1	6
Savarankiškas aparatūrinės įrangos simuliavimas	3	7,5	2	5,5
Programinis architektūros simuliavimas	2	7	2	5
Pirmoji siūloma alternatyva	3	6	1	4
Supaprastinta architektūra	3	8	4	3
Egzistuojančios sistemos modifikavimas	1	3	1	3



1 pav. – mokymo metodikų kriterijų įverčių balais diagrama

3 Medžiaga, remianti siūlomą metodiką

Šio skyriaus paskirtis pateikti medžiagą, kuri reikalinga siūlomos praktinių užsiėmimų metodikos alternatyvos įgyvendinimui. Norint tai atlikti, reikia:

1. pateikti operacinės sistemos kūrimo aplinkos, leidžiančios atlikti visas reikalingas kompiliavimo, transliavimo ir kitas operacijas, aprašymą;
2. pateikti teorinę medžiagą ir pavyzdžius, susijusius su operacinės sistemos aparatūrinės dalies veikimo principais.

Šiai užduočiai atlikti, reikėjo sudaryti operacinės sistemos kūrimo aplinką sudarančių įrankių komplektą, kuris atlikus analizę parodė geriausius rezultatus. Taip pat, atrinkti ir pateikti informaciją apie aparatūrinės įrangos įtaką sistemos kūrimui, pavyzdžiui, kaip vyksta duomenų įvedimas klaviatūra.

Dėmesys sutelkiamas į praktinę sistemų kūrimo pusę. Tam pasitelkiamas MyOS projektas, kuris susideda iš pritaikytos sistemos kūrimui aplinkos ir kiekviename skyriuje papildomos pavyzdinės programos.

3.1 Operacinės sistemos kūrimo aplinka

3.1.1 Įrankių pasirinkimo kriterijai

Šiuo projektu siekiama pateikti patogią ir paprastą darbo aplinką, skirtą kurti ir tobulinti operacinę sistemą. Tai pasiekiama ne tik pasitelkus tinkamus įrankius, bet ir viską tinkamai suderinus. Sistema kuriama Intel x86 architektūrai, todėl įrankiai parinkti jai. Taigi, kuriant operacinę sistemą kitai architektūrai šie įrankiai gali netikti.

Kriterijai (svarbos mažėjimo tvarka):

1. kaina – turi būti nemokamai;
2. prieinamumas – galimybė lengvai gauti įrankį;
3. tarpusavio suderinamumas;
4. ekspertinis vertinimas;
5. paplitimas – įrankis populiarus.

Projektui buvo parenkami tokie įrankiai, kuriuos galima naudoti nemokamai visiems, taigi, kiekvienas sumanęs kurti operacinę sistemą, gali jais pasinaudoti, ir nereikia jų pirkti arba naudotis nelegalia programine įranga. Be to, visus įrankius galima atsisiųsti iš Interneto (atsisiuntimo adresai pateikti skyriuje 3.1.4 Žiniatinklio adresai). Įrankių licencijos: GNU General Public License (< http://www.gnu.org/copyleft/gpl.html >), GNU Lesser General Public

License (< http://www.gnu.org/copyleft/lesser.html >), Eclipse Public License (< http://www.eclipse.org/legal/epl-v10.html >).

3.1.2 OS kūrimui reikalingi įrankiai

OS projektui įgyvendinti parinkti įrankiai yra skirti GNU/Linux operacinei sistemai. Šiuo konkrečiu atveju buvo naudojama Ubuntu. Ubuntu GNU/Linux operacinė sistema yra paremta atviro kodo komponentais ir, kaip ir parinkti įrankiai, yra visiškai nemokamai ir visiems prieinama.

OS kūrimui reikalingi įrankiai:

1. programavimo aplinka;
2. kompiliatorius;
3. adresų transliatorius (angl. linker);
4. paleidimo posistemė (angl. boot loader);
5. įrankiai, leidžiantys keisti diskų atvaizdų (angl. image) failus;
6. Intel x86 architektūros emuliatorius;
7. skriptai.

3.1.2.1 Programavimo aplinka

Akivaizdu, kad kuriant sistemą reikia įrankio, kuriuo būtų galima rašyti sistemos kodą. Žinoma, galima naudotis paprastu teksto redaktoriumi, tačiau kur kas našiau yra naudoti tam skirtą įrankį. Programavimo įrankio parinkimo kriterijai:

- programavimo kalbos raktinių žodžių ryškinimas;
- automatinis raktinių žodžių užbaigimas;
- lengvas projekto failų naršymas;
- lengvas funkcijų kodo pasiekimas ir jų aprašymo pateikimas;
- galimybė kompiliuoti arba paleisti skriptus iš programos bei galimybė keisti paleidimo parametrus.

Vertinant programavimo įrankius norima išsirinkti labiausiai kriterijus atitinkantį įrankį, kuris leidžia patogiai įvykdyti reikiamas operacijas bei nereikalauja atsidaryti papildomų įrankių, langų. Toks įrankis leidžia efektyviai naudoti kūrimo aplinką ir nesiblaškyti.

3.1.2.1.1 C kalba

C programavimo kalbai skirtų nemokamų įrankių yra tikrai nemažai ir įvairioms platformoms, pvz.: Code::Blocks Studio, Anjuta, Dev-C++, Eclipse CDT, Borland C++ 5.5, MinGW, Turbo C. Dėl anksčiau išvardintų kriterijų bei ekspertinio vertinimo šiam projektui įgyvendinti buvo pasirinkta Eclipse CDT programavimo aplinka. Eclipse yra įskiepiais (angl.

plug-in) pagrista programavimo aplinka, leidžianti lengvai keisti ir praplėsti jos galimybes. Eclipse CDT yra Eclipse atmaina, skirta konkrečiai C ir C++ programavimui.

3.1.2.1.2 ASM kalba

Asemblerio programavimo kalbai specialiai skirtų nemokamų įrankių nėra daug, pvz.: Chrome IDE, Easy Code, RadASM, bet daugelis kitų programavimo aplinkų ir netgi kai kurie tekstiniai redaktoriai palaiko šios kalbos raktinių žodžių ryškinimą. Eclipse CDT taip pat palaiko šią funkciją.

3.1.2.1.3 Kiti aspektai

Patogumo dėlei reikia galimybės nesunkiai paleisti reikalingus skriptus iš programavimo aplinkos. Eclipse CDT puikiai integruoja šią galimybę su Make skriptais. Tai pat ši programavimo aplinka leidžia lengvai naršyti projekto failus, visus juos redaguoti, taigi jos pilnai užtenka vykdyti visai projekto eigai ir nereikia atsidaryti papildomų langų.

3.1.2.2 Kompiliatorius

3.1.2.2.1 C kompiliatorius

C kompiliatorius buvo pasirinktas gerai žinomas ir paplitęs C kompiliatorius iš GCC (GNU Compiler Collection) kompiliatorių kolekcijos – gcc.

3.1.2.2.2 Asembleris

Asembleris šiam projektui buvo pasirinktas NASM (The Netwide Assembler).

3.1.2.3 Adresų transliatorius

Šiame projekte naudojamas GNU Binary Utilities paketo adresų transliatorius, Unix operacinės sistemos ld komandos implementacija Linux operacinei sistemai.

3.1.2.4 Paleidimo posistemė

GNU GRUB – Grand Unified Bootloader – viena iš populiariausių GNU/Linux sistemų paleidimo posistemių. Kiti pvz.: LILO, Smart Boot Loader, XOSL Boot Loader, GAG The Graphical Boot Loader. GRUB paleidimo posistemė yra pakankamai aukšto lygio funkciniu požiūriu ir labai universalus. Jis gali būti įdiegtas tiek į pradinį įkrovos takelį (angl. MBR), tiek į kietojo disko dalį, leidžia užkrauti beveik bet kokią operacinę sistemą, palaiko daugelį failų sistemų ir jo konfigūracija gali būti keičiama po diegimo. GRUB skiriasi nuo kitų paleidimo posistemių tuo, kad pateikia naudotojui komandinės eilutės naudotojo sąsają, kuri leidžia nustatyti užkrovimo parametrus kompiuterio paleidimo metu.

Paleidžiant kompiuterį, iš pradinio įkrovos takelio užkraunama GRUB pirma pakopa (angl. stage 1). Kadangi pradinio įkrovos takelio dydis yra mažas, pirma pakopa tiesiog užkrauna

GRUB antrą pakopą (angl. stage 2), kuri gali būti bet kurioje disko vietoje. Kai antra pakopa gauna valdymą, ji pateikia naudotojo sąsają ir naudotojas gali pasirinkti norimą operacinę sistemą.

3.1.2.5 Įrankiai, leidžiantys keisti disko atvaizdų failus

GNU/Linux operacinė sistema aprūpinta komandomis, įgalinančiomis dirbti su diskų atvaizdais. Deja, joms reikia supervartotojo teisių, todėl jos bus naudojamos tik pirmam atvaizdui sukurti. Naudojamos komandos:

losetup, losetup –d, mount, umount

Paprasto vartotojo režime naudojama komandą iš Mtools įrankių rinkinio:

mcopy

Ji leidžia kopijuoti failus tiesiai iš failų sistemos į disko atvaizdą (bet tik msdos failų sistemą).

3.1.2.6 Intel x86 architektūros emuliatorius

Smarkiai palengvinantis operacinių sistemų kūrimą yra pasirinktos architektūros emuliatorius, šiuo atveju – Intel x86. Projektui buvo pasirinktas nemokamai Bochs emuliatorius, palaikantis procesorių, atminties, diskų, ekrano, tinklo, bazinės įvesties/išvesties sistemos (angl. BIOS) bei kitų standartinių personalinių kompiuterio elementų emuliaciją. Emuliatorius plačiai naudojamas operacinių sistemų kūrėjų mėgėjų, nes skirtingai nei daugelis kitų emuliatorių palaiko klaidų pranešimus ir galimybę išsaugoti veiklos būsenas failuose (angl. dump files).

3.1.2.7 Skriptai

Norint našiai dirbti, reikia nuolatos pasikartojančias užduotis ar jų sekas automatizuoti. Kuo daugiau tokių užduočių automatizuojama, tuo labiau galima susikaupti prie svarbiausio dalyko – operacinės sistemos kodo rašymo. Pasitelkus GNU Make skriptus galima automatizuoti visas operacinių sistemų kūrimo užduotis, pradedant kompiliavimu ir baigiant emuliatoriaus paleidimu. Skriptą galima paleisti iš konsolės arba integruoti į Eclipse CDT grafinę sąsaja ir taip dar patogiau juo naudotis.

3.1.3 Įrankių diegimas ir konfigūravimas

3.1.3.1 Pastabos

Keletas pastebėjimų apie GNU/Linux sistemą bei naudojimąsi ja. Visų pirma, GNU/Linux operacinėje sistemoje yra skirtumas tarp mažųjų bei didžiųjų raidžių. Taigi naudojant komandas, katalogų bei failų pavadinimus reikia į tai atsižvelgti.

Visas komandas galima vykdyti konsolėje. Konsolę (angl. terminal) galima pasileisti iš menu arba naudoti ją neužkrovus grafinės aplinkos.

Viena iš pagrindinių komandų konsolėje yra „cd“. Ja naudojantis galima „nueiti“ į norimą katalogą. Naudojimas:

cd *katalogo_pavadinimas*

Komanda

sudo *programos_pavadinimas*

suteikia programai nurodytai **[programos_pavadinimas]** supervartotojo teises Ubuntu operacinėje sistemoje. Komanda gali pareikalauti naudotojo slaptažodžio.

Komanda

aptitude install *pavadinimas*

iškviečia Ubuntu operacinės sistemos paketų tvarkyklę ir įdiegia programą, paketą arba biblioteką nurodyta **[pavadinimas]**.

3.1.3.2 Naudojami įrankiai

Šiame poskyryje išvardinti šio projekto metu naudoti įrankiai ir jų tikslios versijos. Naudota operacinė sistema Ubuntu 8.10 (taip pat išbandyta su 8.04, 9.04).

- Eclipse CDT (versija: 3.4.2);
- Bochs x86 Emulator (versija: 2.3.7);
- GNU Make (versija: 3.81);
- GCC (versija: 4.3.2);
- GNU ld (versija: 2.18.93);
- Mtools (versija 3.9.11);
- Nasm (versija: 2.03.01).

3.1.3.3 Projekto katalogų medis

Aiškumo dėlei apibrėžiamas projekto failų bei katalogų išdėstymas. Taip lengviau bus orientuotis gilinantis į įrankių diegimą bei konfigūraciją.

[~/] – pagrindinis katalogas, kuriame įdėti visi projekto failai. Tai gali būti bet koks katalogas, tačiau patartina projektą kurti naudotojo namų arba kitoj lengvai pasiekiamoje direktorijoje. Taip lengviau bus įvykdyti komandas konsolėje.

[~/eclipse_workspace/] – katalogas, kuriame kuriami Eclipse CDT projektai (šiuo atveju tik vienas).

[~/eclipse_workspace/.metadata/] – šiame kataloge saugomi Eclipse CDT metaduomenys.

[~/eclipse_workspace/MyOS/] – pagrindinis naujos operacinės sistemos katalogas. Jame saugomi failai:

[bochrc] – Bochs emuliatoriaus nustatymai;

[kernel.bin] – sukompiliuotas kernelis;

[link.ld] – adresų transliatoriaus skriptas;

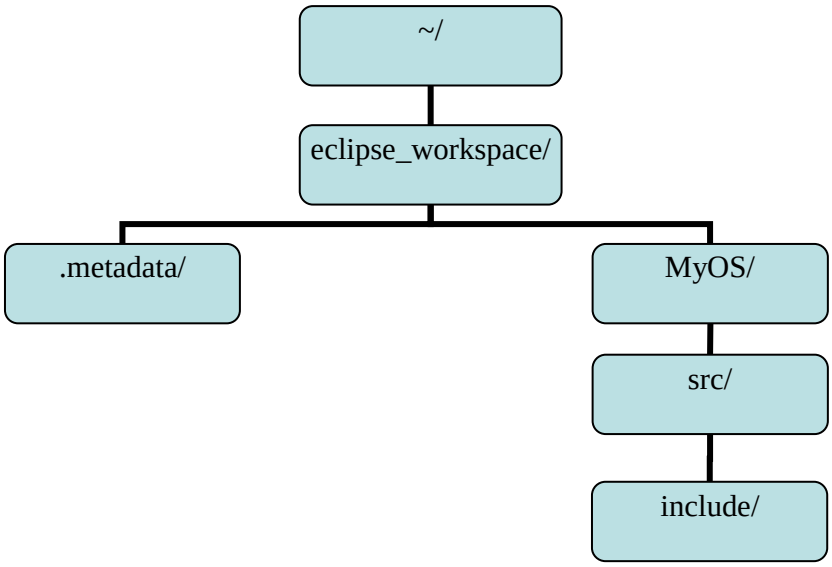
[makefile] – Make skriptas;

[myos.img] – disko atvaizdas, saugantis kuriamą operacinę sistemą, skirtas emuliatoriaus paleidimui;

[.cproject] ir **[.project]** – Eclipse CDT programos konfigūraciniai failai.

[~/eclipse_workspace/MyOS/src/] – kodo failų katalogas. Jame saugomi visi **[*.c]** bei **[*.asm]** failai.

[~/eclipse_workspace/MyOS/src/include/] – C kalbos antraštės failai (**[*.h]**).



Figūra 1 – Katalogų medis

Katalogai **[eclipse_workspace]** ir **[MyOS]** gali būti pavadinti kitaip, bet reikia į tai atsižvelgti sekant konfigūracijos instrukcijas.

3.1.3.4 GCC, GNU Make, ld

GNU Toolchain paketas yra pagal nutylėjimą įdiegtas Ubuntu sistemoje bei daugelyje kitų GNU/Linux operacinių sistemų. GCC, GNU Make ir ld yra šio paketo dalis, taigi nieko papildomai diegti nereikia.

3.1.3.5 Bochs x86 Emulator

3.1.3.5.1 Diegimas Ubuntu aplinkoje

Konsolėje įvykdyti komandą:

```
sudo aptitude install bochs bochs-x
```

3.1.3.5.2 Konfigūracija

Emuliatorius įdiegtas ir jau gali būti paleistas, tačiau reikia sukurti konfigūracijos failą, skirtą kuriamai operacinei sistemai, kad kiekvieną kartą paleidus emuliatorių nereikėtų iš naujo keisti nustatymų.

Konsolėje įvykdyti komandas:

```
cd ~/eclipse_workspace/MyOS/
bochs
```

Pasileidžia Bochs emuliatoriaus konfigūracijos meniu. Komandos vykdomos pasirinkus menu punktų skaičius (nurodyta skliausteliuose) arba įvedant reikiamus parametrus (nurodyta kabutėse, įvesti be jų).

1. rinktis „Edit options“ (3);
2. rinktis „Disk options“ (10);
3. rinktis „Floppy Disk 0“ (1);
4. įvesti „myos.img“;
5. įvesti „auto“;
6. rinktis nustatytą reikšmę (enter);
7. rinktis „Boot Options“ (15);
8. rinktis „Boot drive #1: floppy“ (1);
9. įvesti „floppy“;
10. grįžti meniu atgal (0) tris kartus;
11. rinktis „Save options to...“ (4);
12. įvesti „bochsrc“;
13. rinktis „Quit now“ (7).

*Pastaba. Emuliatorius nurodyta disko atvaizdą **[myos.img]** laikys esant diskelių įrenginiu (angl. floppy) ir iš jo pirmiausia bandys užkrauti operacinę sistemą.*

Bochs emuliatorius sukonfigūruotas. Dabar jį paleidus, automatiškai bus nuskaitytas **[bochsrc]** failą ir, jei **[myos.img]** turi įrašytą operacinę sistemą, ją užkraus. Emuliatorius paleidžiamas iš konsolės tokia komanda:

```
bochs –q
```

Reikia pastebėti, kad reikia prieš tai „nueiti“ į „~/eclipse_workspace/MyOS/“ katalogą. Tai galima padaryti įvykdžius komandą:

```
cd ~/eclipse_workspace/MyOS/
```

3.1.3.6 GRUB paleidimo posistemė

Pagal nutylėjimą Ubuntu operacinė sistema naudoje GRUB paleidimo posistemę, taigi nieko papildomai diegti nereikia. Paleidimo bei konfigūracijos failai gali būti randami **[/boot/grub/]** kataloge.

3.1.3.7 Įrankiai, leidžiantys keisti disko atvaizdų failus

3.1.3.7.1 Mtools diegimas

Didelė tikimybė, kad Mtools bus jau įdiegtas Ubuntu sistemoje. Tačiau jei taip nėra, įdiegti galima įvykdžius komandą:

```
sudo aptitude install mtools
```

3.1.3.7.2 Operacinės sistemos disko atvaizdo sukūrimas

Čia aprašoma operacinės sistemos disko atvaizdo sukūrimas. Atvaizdas vėliau bus panaudotas paleidžiant emuliatorių ir išbandant sukompiliuotą branduolį. Tai reikia atlikti tik vieną kartą pradžioje. Vėliau yra keičiamas tik atnaujintas branduolys. Šioms procedūroms reikia supervartojo teisių, taigi reikia susirasti kompiuterį kuriame galima vykdyti komandas supervartotojo teisėmis arba kreiptis į kompiuterių administratorius pagalbos. Taip pat galima susirasti jau padarytą disko atvaizdį.

Projekto kataloge (**[~/eclipse_workspace/MyOS]**) sukuriamas katalogas **[boot/grub]**. Į šį katalogą reikia įdėti du GRUB paleidimo posistemės failus: **[stage1]** ir **[stage2]**. Juos galima parsisiųsti iš GRUB namų puslapio arba nukopijuoti iš esamos sistemos[1]. Ubuntu operacinėje sistemoje galima konsolėje įvykdyti komandas:

```
cd ~/eclipse_workspace/MyOS/
cp /boot/grub/stage1 boot/grub
cp /boot/grub/stage2 boot/grub
```

Tame pačiame kataloge reikia sukurti failą pavadinimu **[menu.lst]**. Jame įrašome:

```
timeout 1
title= MyOS
root (fd0)
kernel /kernel.bin
```

Visi failai paruošti, laikas sukurti ir paruošti disko atvaizdą. Įvykdome:

```
cd ~/eclipse_workspace/MyOS/
```

[1] Reikalingi failai taip pat gali būti rasti projekto CD [extra/boot/] kataloge.

```
dd if=/dev/zero of=myos.img bs=512 count=2880
sudo losetup /dev/loop0 myos.img
sudo mkfs.msdos /dev/loop0
sudo mount /dev/loop0 /mnt/
sudo cp boot –R /mnt/
sudo umount /mnt/
sudo grub –device-map\=/dev/null
    device (fd0) /dev/loop0
    root (fd0)
    setup (fd0)
    quit
sudo losetup –d /dev/loop0
```

Įvykdę šias komandas jau turime paruoštą disko atvaizdą su įdiegta GRUB paleidimo posisteme. Jei Bochs emuliatorius jau sukonfigūruotas, galima išbandyti jį paleisdami.

Kiekviena kartą atnaujinus branduolį, jį reikia iš naujo įkelti į disko atvaizdą. Tai galime padaryti pasinaudoję komanda:

```
mcopy –i myos.img kernel.bin ::/ -v –Do
```

Šiai komandai užtenka paprasto vartotojo teisių.

3.1.3.8 Eclipse CDT

3.1.3.8.1 Diegimas Ubuntu aplinkoje

Parsisiunčiamas programos paketą iš <http://www.eclipse.org/downloads/>. Puslapyje pasirenkama Eclipse CDT. Rašymo metu tiesioginis adresas yra <http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/ganymede/SR2/eclipse-cpp-ganymede-SR2-linux-gtk.tar.gz>. Išarchyvuojamas parsiųstą failą į pasirinktą katalogą, pavyzdžiui: **[~/programs/]**.

Įdiegiamas Sun Java paketas, jei neįdiegtas. Komanda:

```
sudo aptitude install sun-java5-jre
```

Programa galima paleisti iš konsolės:

```
cd ~/programs/eclipse/
./eclipse &
```

arba

```
~/programs/eclipse/eclipse &
```

Tačiau daug patogiau pasidaryti nuorodą, kad būtų galima lengvai pelės paspaudimu paleisti programą.

3.1.3.8.2 Konfigūracija

Paleidus Eclipse CDT pirmą kartą, paklausiama darbinio katalogo vietos. Nurodomas „~/eclipse_workspace“. Sukuriamas naują projektą. Reikia pasirenkame „File“ -> „New“ -> „C Project“. Įrašomas naujo projekto pavadinimą „MyOS“ ir spaudžiama „Finish“.

~/eclipse_workspace/MyOS/ kataloge sukuriamas naują failą pavadinimu **[makefile]** ir du naujus katalogus: **[src]**, o jo viduje – **[include]**. Atsidaromas **[makefile]** ir įrašoma:

```
debug:
    echo „Make Debug“
release:
    echo „Make Release“
emulate:
    echo „Make Emulate“
```

Šį failą vėliau bus pakeistas, kad jis iš tikrųjų kompiliuotų projektą. Kol kas jis tik padės sukonfigūruoti Eclipse CDT. Svarbu, kad būtų išlaikytas eilučių lygiavimas šiame faile.

Atidaroma „*Project*“ -> „*Properties*“. Parenkama „*C/C++ Build*“. Ten parenkama „*Manage Configurations...*“. Sukuriama nauja konfigūracija ir pavadinama „*Emulate*“. Uždarome langą. Nustatoma (jei nustatyta kitaip) „*Debug*“ konfigūracija. „*Builder Settings*“ lange pakeičiama „*Build directory*“ į „*${workspace_loc:/MyOS/}*“. Nuimamas pažymėjimas nuo „*Use default build command*“. „*Build command*“ pakeičiama į „*make debug*“. „*Behavior*“ lange pažymimas „*Build on resource save (Auto build)*“ ir ištinama reikšmė. Taip pat ištrinama ir „*Build (Incremental build)*“ reikšmė.

Tas pats padaroma ir kitom dviem konfigūracijoms („*Release*“ ir „*Emulate*“), tik atitinkamai vietoj „*debug*“ naudojame „*release*“ ir „*emulate*“. Skirtingai tik „*Emulate*“ konfigūracijoje nepažymime „*Build on resource save (Auto build)*“.

Įvykdę šiuos žingsnius galima naudoti make skriptą iš Eclipse CDT grafinės sąsajos mygtuku arba meniu pagalba. Be to, jei programos kodo konstravimo (angl. build) režimas įjungtas į „*Debug*“ arba „*Release*“, išsaugojus pakeitimus bet kuriame faile, Eclipse CDT automatiškai paleidžia make skriptą ir iš naujo sukompiliuojami kodo išeities failai arba parodomos klaidos.

3.1.3.9 Skriptai

3.1.3.9.1 Makefile

Make skriptas (**[makefile]**) atlieka pagrindinį kompiliavimo, adresų transliavimo bei paleidimo darbą. Šiame faile aprašytos trys taisyklės:

1. **debug** – sukompiliuoja visus **[*.c]** ir **[*.asm]** failus ir sutransliuoja į **[kernel.bin]**;
2. **release** – įvykdo **debug** taisyklę ir nukopijuoja **[kernel.bin]** į **[myos.img]** disko atvaizdą;
3. **emulate** – įvykdo **release** ir paleidžia Bochs emuliatorių.

Šis skriptas parašytas taip, kad automatiškai rastų visus **[*.c]** ir **[*.asm]** failus **[~/eclipse_workspace/MyOS/src]** kataloge, sukurtų objektų failus ir juos sutransliuotų į **[kernel.bin]**. Po kiekvienos transliacijos objektų failai yra pašalinami. Šio failo turėtų pilnai pakakti operacinės sistemos įgyvendinimui. Jei vis dėl to prisireiktų jį keisti, make skriptų dokumentacija yra pateikta internetiniame puslapyje adresu <http://www.gnu.org/software/autoconf/manual/make/index.html>.

```
Cflags := -Wall -O -fstrength-reduce -fomit-frame-pointer\
	-finline-functions -nostdinc -fno-builtin -I./src/include -c
ASMFlags := -f aout

sourceC := $(wildcard ./src/*.c)
sourceASM := $(wildcard ./src/*.asm)
objects := $(patsubst ./src/%,%,$(patsubst %.c,%.o,$(sourceC))\
	$(patsubst %.asm,%.o,$(sourceASM)))

debug: $(objects) kernel.bin clean

release: debug create_img

emulate: release
	bochs -q &

kernel.bin: $(objects)
	ld -T link.ld -o kernel.bin $(objects)

create_img:
	mcopy -i myos.img kernel.bin ::/ -v -Do

clean:
	rm *.o

%.o : ./src/%.c
	gcc $(Cflags) $< -o $@
```

```
%.o : ./src/%.asm
	nasm $(ASMFlags) $< -o $@
```

1 tekstas – make skriptas [makefile]

Pastabos:

- *Cflags – C kompiliatoriaus parametrai.*
- *ASMFlags – asemblerio parametrai.*
- *Tuščios eilutės įterptos norint palengvinti skaitymą.*
- *„tab“ simboliai yra būtini taisyklės antroje eilutėje.*
- *Pasvirasis kairinis brūkšnys („\“) naudojamas ilgos eilutės rašymui į kelias norint palengvinti skaitymą. Make skripte eilutė besibaigianti pasviruoju brūkšniu bei eilutė po ją traktuojama kaip viena.*

3.1.3.9.2 Adresų transliatoriaus (ld) skriptas

Adresų transliatorius yra įrankis, paimantis visus C kompiliatoriaus ir asemblerio išeities failus ir juos sutransliuojantis į vieną vykdomąjį failą. Vykdomasis failas gali būti kelių formatų. Labiausiai naudojami: Flat, AOUT, COFF, PE, ELF. Pasirinktam adresų transliatoriui – ld – reikia pateikti skriptą, pagal kurį jis transliuos objektų failus į **[kernel.bin]**.

```
OUTPUT_FORMAT(„binary“)
ENTRY(start)
phys = 0x00100000;
SECTIONS
{
  .text phys : AT(phys) {
    code = .;
    *(.text)
    *(.rodata)
    . = ALIGN(4096);
  }
  .data : AT(phys + (data – code))
  {
    data = .;
    *(.data)
    . = ALIGN(4096);
  }
  .bss : AT(phys + (bss – code))
  {
    bss = .;
```

```
        *(.bss)
        . = ALIGN(4096);
      }
      end = .;
    }
```

2 tekstas – adresų transliatoriaus skriptas [link.ld]

*Pastaba. Eilutėje „**ENTRY**(start)“ žodis „start“ nurodo nuo kurio failo bus pradėta transliacija. Šiuo atveju **[start.asm]** asembleris sukompiliuos į **[start.o]**, ir nuo jo transliaciją pradės ld adresų transliatorius.*

Informacijos apie ld skriptų sintaksę galima rasti internete adresu <http://sources.redhat.com/binutils/docs-2.16/ld/Scripts.html#Scripts>.

3.1.3.10 Operacinės sistemos paleidimas iš usb atmintinės

Patogu, kuriant operacinę sistemą, ją išbandyti pasinaudojus emuliatoriumi: nereikia perkrauti kompiuterio kiekvieną kartą padarius pakeitimus, galima greitai juos patikrinti. Tačiau sistema yra kuriama realiai architektūrai, todėl anksčiau ar vėliau ją reikia paleisti tikrame kompiuteryje. Seniau tam dažniausiai buvo naudojami diskeliai (angl. floppy disk), tačiau dabar daug labiau paplitusios yra usb atmintinės.

Svarbiausia ir sudėtingiausia dalis yra įrašyti GRUB paleidimo posistemę į usb atmintinę. Pradžiai reikia usb atmintinės su GRUB palaikoma failų sistema. Kadangi ši paleidimo posistemė palaiko labai daug failų sistemų, tai labai maža tikimybė, kad pasirinkta failų sistema bus netinkama. Rasti palaikomų failų sistemų sąrašą galima per internetinį GRUB namų puslapį. Usb atmintinėje reikia sukurti katalogą **[/boot/grub/]**. Į jį nukopijuoti **[stage1]** ir **[stage2]**. Kur rasti šiuos failus reikia pasitikslinti skyriuje 3.1.3.7.2 Operacinės sistemos disko atvaizdo sukūrimas.

Iš konsolės paleidžiama:

```
sudo grub
```

Toliau vykdomos komandos GRUB konsolėje.

```
find /boot/grub/stage1
```

Programa pateiks diskų ir jų skirsnių sąrašą, kurie turi reikimą failą. Sąrašo pavyzdys:

```
(hd0,1)
(hd0,4)
(hd2,0)
```

Svarbu tiksliai nustatyti kuris diskas iš pateiktų yra usb atmintinė. Didžiausia tikimybė, kad jis bus paskutinis pateiktas sąraše. Taip pat skirsnio numeris turėtų būti 0 (hd***x***,0). Toliau vykdoma komandų seka (***x*** pakeičiama į konkretaus disko numerį):

```
root (hdx,0)
setup (hdx)
quit
```

Paleidimo posistemė įdiegta. Reikia sukurti konfigūracijos failą **[menu.lst]** kataloge **[/boot/grub/]**. Jame įrašoma:

```
timeout 1
title= MyOS
root (hd0,0)
kernel /kernel.bin
```

GRUB paleidimo posistemė jau beveik paruošta. Pradžiai dar reikia įkelti sukompiliuotą branduolio atvaizdą **[kernel.bin]** į usb atmintinę. Nustatoma, kad kompiuteris pirmiausia krautų operacinę sistemą iš usb atmintinės ir paleidžiamas kompiuteris. Toliau viskas priklauso nuo kompiuterio bazinės įvesties/išvesties sistemos (angl. basic input/output system) realizacijos. Skirtingi kompiuteriai skirtingai nustato kuris diskas yra pagrindinis kompiuterio paleidimo metu. Be to, kai kurie kompiuteriai usb atmintines laiko diskelių įrenginiais. Jei sistema užsikrauna, tai viskas yra gerai ir nieko keisti nereikia. Jei ne – reikia nustatyti kitą diską iš kurio GRUB paleidimo posistemė bando užkrauti operacinę sistemą. GRUB menu lange paspaudus „e“ pasiekiame parametrų langą (tie patys parametrai, kurie buvo aprašyti **[menu.lst]** faile). Pažymime juostą pirmą juostą („root (hd0,0)“). Vėl reikia spausti „e“. Patenkama į keitimo režimą. Čia reikia nustatyti kitą diską. Nutrynus eilutės galą, palikus tik „root (hd“ ir paspaudus „tab“ pateikiamas visų diskų sąrašas. Pasirenkamas kitą iš eilės („root (hd1,0)“) ir paspaudus „enter“ grįžtama į parametrų langą. Dabar paspaudus „b“ bandoma užkrauti operacinę sistemą. Jei nepavyksta vėl jos užkrauti, reikia kartoti aprašytą procedūrą ir nurodyti kitą diską. Atradus tinkamą diską verta pakoreguoti **[menu.lst]** failą.

Usb atmintinė paruošta, norint išbandyti naują branduolį tereikia jį nukopijuoti vietoj senojo ir užkrauti jį iš atmintinės.

3.1.4 Žiniatinklio adresai

Šiame skyriuje pateikiami visi esminiai internetiniai adresai, susiję su anksčiau aprašytais įrankiais. Savaime suprantama, su laiku galimi svetainių/adresų pakeitimai, tačiau bent namų adresai turėtų nesikeisti, o juose nesunku rasti nuorodas parsisiuntimui bei diegimo instrukcijas.

3.1.4.1 Eclipse CDT

- namų adresas: < http://www.eclipse.org/ >;
- adresas parsisiųsti:
  <http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/ganymede/SR2/eclipse-cpp-ganymede-SR2-linux-gtk.tar.gz >.

3.1.4.2 GNU GCC

- namų adresas: < http://gcc.gnu.org/ >;
- informacija apie naudojimą: < http://linux.die.net/man/1/gcc >;
- adresas parsisiųsti: < http://gcc.gnu.org/install/binaries.html >.

3.1.4.3 NASM

- namų adresas: < www.nasm.us >;
- adresas parsisiųsti: < http://www.nasm.us/pub/nasm/releasebuilds/?C=M;O=D >.

3.1.4.4 Make

- namų adresas: < http://www.gnu.org/software/make/ >;
- informacija apie skriptų sintaksę:
  < http://www.gnu.org/software/autoconf/manual/make/index.html >;
- adresas parsisiųsti: < http://ftp.gnu.org/pub/gnu/ >.

3.1.4.5 Ld

- namų adresas: < http://www.gnu.org/software/binutils/ >;
- informacija apie naudojimą: < http://linux.die.net/man/1/ld >;
- skriptų sintaksė:
  < http://sources.redhat.com/binutils/docs-2.16/ld/Scripts.html#Scripts >;
- adresas parsisiųsti: < http://ftp.gnu.org/gnu/binutils >.

3.1.4.6 GRUB

- namų adresas: < http://www.gnu.org/software/grub/ >;
- adresas parsisiųsti: < ftp://alpha.gnu.org/gnu/grub/ >;
- informacija apie konfigūravimą:
  < http://www.gnu.org/software/grub/manual/grub.html#Configuration >.

3.1.4.7 Bochs x86 Emulator

- namų adresas: < bochs.sourceforge.net >;

- vartotojo instrukcija:

  < http://bochs.sourceforge.net/doc/docbook/user/index.html >;

- adresas parsisiųsti:

  <http://bochs.sourceforge.net/cgibin/topper.pl?name=See+All+Releases&url=http://sourceforge.net/project/showfiles.phpqmrkgroup_ideq12580 >.

3.1.4.8 Mtools

- namų adresas: < http://www.gnu.org/software/mtools/intro.html >;
- adresas parsisiųsti: < http://www.gnu.org/software/mtools/download.html >.

3.2 Operacinių sistemų sąveika su aparatūrine įranga

Skyriuje pateikiama teorinė informacija apie aparatūrinės įrangos įtaką operacinei sistemai. Praktinį medžiagos pritaikymą iliustruoja nuolat papildoma MyOS pavyzdinė programa. Tokiu būdu viską galima suskirstyti į tarpusavyje nepriklausomas dalis, tačiau kartu parodyti, kaip jos sąveikauja tarpusavyje, pavyzdžiui, pertraukimų deskriptorių lentelės įrašymas ir pertraukimus apdorojančių paprogramių panaudojimas. Pateikiama tokia informacija, kuri sudaro pagrindines funkcijas gebantį atlikti branduolį (angl. kernel). Taigi, MyOS yra mažas sistemos branduolys.

Skyrius prasideda nuo branduolio sąvokos ir jo galimybių apžvalgos. Vėliau nuosekliai pristatoma kiekviena jo savybė, pateikiant informaciją apie įrangos įtaką sistemai ir praktinį žinių pritaikymą.

3.2.1 Branduolio sąvoka

Branduolys yra operacinės sistemos dalis, kuri atsakinga už kompiuterio resursų valdymą: procesoriaus, atminties, pertraukimų ir kitų. Jis tarsi tarpininkas tarp aparatūrinės įrangos ir kitų sistemos dalių.

3.2.2 MyOS branduolio apžvalga

Kaip jau minėta, pateikiama pavyzdinė programa yra bazines funkcijas galintis atlikti branduolys. Atliekamų funkcijų sąrašas:

1. duomenų išvedimas į ekraną;
2. globalių deskriptorių lentelės (GDL, angl. global descriptor table) užpildymas;
3. pertraukimų deskriptorių lentelės (PDL, angl. interrupt descriptor table) užpildymas;
4. pertraukimų apdorojimo paprogramių (PAP, angl. interrupt services routine) nustatymas, siekiant valdyti pertraukimus ir pertraukimų užklausas (PU, angl. interrupt request);
5. programuojamų pertraukimų valdiklių (PPV, angl. programmable interrupt controller) perprogramavimas į naujas PDL reikšmes;
6. PU įdiegimas ir aptarnavimas;
7. programuojamo intervalo taimerio valdymas;
8. klaviatūros valdymas ir duomenų skaitymas.

Kiekviena iš aukščiau išvardintų savybių pristatoma atskirame skyriuje, pateikiant teorinę medžiagą apie aparatūrinės įrangos veikimą ir jos pritaikymą MyOS projekte.

3.3 Pagalbinės funkcijos

Prieš pradedant programuoti konkrečias branduolio savybes, reikia apibrėžti bazines funkcijas, kurios pagelbės kuriant įvairų funkcionalumą. Failas [**main.c**]:

```
#include <system.h>
/* Nukopijuoja „count“ kiekį baitų iš src į dest */
unsigned char *memcpy(unsigned char *dest, const unsigned char *src, int count)
{
    /* Pridėti kodą, kuris nukopijuotų „count“ kiekį baitų duomenų iš „src“ į
    *  „dest“, galiausiai grąžinti „dest“ */
}
/* Nustato „count“ kiekį baitų adresu „dest“ į „val“ reikšmę */
unsigned char *memset(unsigned char *dest, unsigned char val, int count)
{
    /* Pridėti kodą, kursis nustato „count“ kiekį baitų esantį
    *  „dest“ adresu. Pabaigoje gražinti „dest“ */
}
/* Identiška viršuj esančiai funkcijai, tik čia dirbama su 16
 * bitų reikšmėmis */
unsigned short *memsetw(unsigned short *dest, unsigned short val, int count)
{
    /* Pridėti identišką kodą, esanačiam viršuje, tačiau
     * dirbti su „unsigned short“ tipo kintamaisiais */
}
/* Grąžina eilutės ilgį. Eilutės pabaigos simbolis „\0“ */
int strlen(const char *str)
{
    /* Pridėti kodą, kuris pereina per simbolių masyvą ieškodamas
     * pabaigos simbolio „\0“. Grąžinti rastų simbolių skaičių */
}
/* Funkcija nuskaitanti duomenis iš įvesties/išvesties prievado
 * nurodytu adresu. Joje pasitelkiame C kalbos galimybę iškviesti
 * asemblerio funkcijas. */
unsigned char inportb (unsigned short _port)
{
    unsigned char rv;
    __asm__ __volatile__ ("inb %1, %0" : "=a" (rv) : "dN" (_port));
    return rv;
}
/* Įrašo duomenis į įvesties/išvesties prievadą nurodytu adresu */
void outportb (unsigned short _port, unsigned char _data)
```

```
    {
        __asm__ __volatile__ ("outb %1, %0" : : "dN" (_port), "a" (_data));
    }
    void main()
    {
        /* Vėliau čia reiks pridėti įvairias komandas */
        /* Begalinis ciklas sistemos vykdymui */
        for (;;);
    }
```

1 C kodo ištrauka – [main.c]

Jei nepavyksta suprogramuoti praleistų kodo vietų, visuomet galima kompaktiniame diske pažiūrėti baigtos pavyzdinės programos kodą.

Kadangi visos šios funkcijos bus kviečiamos įvairiuose kituose C kalba parašytuose failuose, reikia jų prototipus aprašyti kitame faile – [**system.h**]:

```
    #ifndef __SYSTEM_H
    #define __SYSTEM_H
    extern unsigned char *memcpy(unsigned char *dest, const unsigned char *src, int
count);
    extern unsigned char *memset(unsigned char *dest, unsigned char val, int
count);
    extern unsigned short *memsetw(unsigned short *dest, unsigned short val, int
count);
    extern int strlen(const char *str);
    extern unsigned char inportb (unsigned short _port);
    extern void outportb (unsigned short _port, unsigned char _data);
    #endif
```

2 C kodo ištrauka – [system.h]

Dabar reikia failus [**main.c**], [**system.h**] ir [**start.asm**] sukompiliuoti ir sutransliuoti į [**kernel.bin**] – štai ir sukurtas paprastas branduolys.

3.4 Duomenų išvedimas į ekraną

Šiame skyriuje detaliai paaiškinta, kokiu būdu išvesti duomenis į ekraną. Tai apima teksto simbolių atvaizdavimą, teksto spalvų nurodymą ir ekrano slinkimo galimybes.

Iš tiesų tai nėra sudėtinga, nes galima naudotis Video Grafikos Komplekto (VGK, video graphics array) plokštės galimybėmis. VGK reiškia standartą, kurį palaiko vaizdo plokštės, dar prieš įrašant tvarkykles. Šis standartas suteikia atminties gabaliuką iš dviejų baitų – atributo ir

simbolio. VGK valdiklis pats pasirūpina simbolių piešimu ekrane. Tuo tarpi ekrano slinkimą aprašyti tenka programuotojui.

Tekstinė atmintis yra tiesiog atminties dalis, kurią galime tiesiogiai pasiekti. Fizinėje atmintyje buferio adresas yra 0xB8000. Buferio tipas yra „short“ tipo, tai reiškia, kad susideda iš 16 bitų masyvo, kurį galima suskaidyti į viršutinius 8 bitus ir apatinius 8 bitus. Pastarieji nurodo vaizdo valdikliui kokį simbolį nupiešti ekrane. Viršutiniai 8 bitai naudojami teksto ir fono spalvoms nurodyti.

2 lentelė – tekstinės atminties struktūra

15 12	11 8	7 0
Fono spalva	Teksto spalva	Simbolis

Taigi, pirmasis baitas skirtas spalvoms vadinamas atributo baitu, o kitas – simbolio. Kadangi kiekvieną spalvą apibrėžia 4 bitai, tai galime naudoti tik 16 spalvų ($2^4 = 16$). Žemiau pateikta spalvų lentelė.

3 lentelė – spalvų paletė

Nr.	Spalva	Nr.	Spalva
0	Juoda (black)	**8**	Tamsiai pilka (dark grey)
1	Mėlyna (blue)	**9**	Šviesiai mėlyna (light blue)
2	Žalia (green)	**10**	Šviesiai žalia (light green)
3	Žalsvai mėlyna (cyan)	**11**	Šviesiai žalsvai mėlyna (light cyan)
4	Raudona (red)	**12**	Šviesiai raudona (light red)
5	Rausvai raudona (Magenta)	**13**	Šviesiai rausvai raudona (light magenta)
6	Ruda (brown)	**14**	Šviesiai ruda (light brown)
7	Šviesiai pilka (light grey)	**15**	Balta (white)

Norint pasiekti konkretų indeksą vaizdo atmintyje, reikia pasitelkti specialią formulę. Teksto atmintis yra linijinė atminties dalis, tačiau vaizdo valdiklis ją naudoja tarsi dvimatį 80x25 masyvą, susidedantį iš 16 bitų reikšmių. Kiekviena reikšmė eina viena paskui kitą. Atsižvelgiant į tai, galima pasitelkti tokią formulę:

```
indeksas = (y_value * width_of_screen) + x_value;
```

Remiantis šia formule, galima nustatyti indeksą tekstinės atminties masyve. Raskime pozicijos (3, 4) indeksą. Pagal formulę 4 * 80 + 3 = 323. Tai reiškia, kad norint nupiešti simbolį ekrano pozicijoje (3, 4), reikia pasitelkti tokį kodą:

```
unsigned short *where = (unsigned short *)0xB8000 + 323; // Pozicija
*where = simbolis | (atributas << 8);                    // Simbolio įrašymas
```

Šia teorija besinaudojantis teksto išvedimas ir ekrano slinkimas yra realizuoti faile **[scrn.c]**. Verta paminėti ekrano poslinkio metodo specifiškumą: paimamas visas tekstas, išskyrus nulinę eilutę. Tuomet jis perkopijuojamas eilute aukščiau. Paskutinė eilutė užpildoma tarpais (22 C kodo ištrauka – [scrn.c]).

Taip pat verta paminėti sudėtingiausią ir didžiausią failo [**scrn.c**] funkciją „putch“, kuri skirta vieno simbolio išvedimui:

```
    /* Išveda vieną simbolį */
    /* Nuoroda į teksto atmintį, teskto ir fono spalvos, x ir y kursorių koordinatės. Gloablūs failo „scrn.c“ kintamieji */
    unsigned short *textmemptr;
    int attrib = 0x0F;
    int csr_x = 0, csr_y = 0;
    void putch(unsigned char c)
    {
        unsigned short *where;
        unsigned att = attrib << 8;
        /* „Backspace“ simbolio valdymas, grįžtam viena pozicija atgal */
        if(c == 0x08)
        {
            if(csr_x != 0) csr_x--;
            where = textmemptr + (csr_y * 80 + csr_x);
            *where = ' ' | att;    /* Simbolis ir atributas - spalva */
        }
        /* Tab simbolio valdymas vykdomas padidinant kursorių x. Bet į
         * tokią poziciją, kuri dalinasi iš 8 */
        else if(c == 0x09)
        {
            csr_x = (csr_x + 8) & ~(8 - 1);
        }
        /* „Carriage Return“ apdorjimas - kursorius grąžinamas į
         * eilutės pradžią */
        else if(c == '\r')
        {
            csr_x = 0;
        }
        /* Nauja eilutė: grįžtam į eilutės pradžią horizontaliai,
         * vertikaliai viena eilute žemyn*/
        else if(c == '\n')
        {
            csr_x = 0;
```

```
            csr_y++;
        }
        /* Simbolių išvedimas */
        else if(c >= ' ')
        {
            where = textmemptr + (csr_y * 80 + csr_x);
            *where = c | att;      /* Simbolis ir atributas - spalva */
            csr_x++;
        }
        /* Jei kursorius pasiekė eilutės pabaigą, tai pereina į naują eilutę */
        if(csr_x >= 80)
        {
            csr_x = 0;
            csr_y++;
        }
        /* Paslenka ekraną, jei reikia ir pajudina kursorių */
        scroll();
        move_csr();
    }
```

3 C kodo ištrauka – [scrn.c]

Aukščiau esančioje kodo ištraukoje panaudotos funkcijos „move_crs“ kodas:

```
    /* Atnaujina kursorių: mirksinti linija ekrane
    *  po paskutinio atspausdinto simbolio */
    void move_csr(void)
    {
        unsigned temp;
        /* Formuė indekso radimui linijinėje atmintyje, išreikšta:
        *  Indeksas = [(y * plotis) + x] */
        temp = csr_y * 80 + csr_x;
      /* Nusiunčia komandą į 14 ir 15 indeksus esančius
        * VGA kontrolerio CRT valdymo registre (Control Register).
        * Tai yra aukštesnieji ir žemesnieji baitai, kurie rodo, kur
        *  kursorius turi mirksėti.
        *  Detaliau:  http://www.brackeen.com/home/vga */
        outportb(0x3D4, 14);
        outportb(0x3D5, temp >> 8);
        outportb(0x3D4, 15);
        outportb(0x3D5, temp);
```

```
    }
```

4 C kodo ištrauka – [scrn.c]

Naudojantis šiomis funkcijomis yra paprasta iš bet kurios funkcijos išvesti duomenis į ekraną. Pavyzdžiui, faile [main.c], funkcijoje „main“ galima parašyti eilutę „putch(‘a‘)“ ir programa išves simbolį „a“ į ekraną. Taip pat, galima parašyti funkciją, kuri naudodamasi „putch“ išvestų į ekraną ištisas teksto eilutes.

3.5 Globalių deskriptorių lentelė

3.5.1 Apibrėžimas

GDL yra duomenų struktūra, kurią naudoja Intel x86 šeimos procesoriai. Jos paskirtis apibrėžti įvairias atminties sričių charakteristikas, vykdant programas. Pavyzdžiui, bazinis atminties adresas, dydis ir priėjimo privilegijos, tokios kaip vykdymas ir rašymas. Šios atminties sritys vadinamos segmentais.

3.5.2 Panaudojimo galimybės

Pasitelkiant į pagalbą GDL galima atlikti šiuos veiksmus:

- apibrėžti atminties segmento privilegijas;
- apibrėžti, ar atminties gabale saugoma vykdoma programa ar duomenys;
- įterpti tokį įrašą į GDL, kad būtų generuojama klaida, kuomet programa bando pasiekti atminties vietą, kuri jai neleistina. Tokiu atveju branduolys gauna galimybes tokios programos veiklą sustabdyti;
- galima apibrėžti užduočių būsenų segmentus (angl. task status segments). Jie naudojami aparatūrine įrangą paremtam multiprogramiškumui;
- nustatyti, ar vykdomas segmentas supervizoriaus ar vartotojo režimu.
- kita.

3.5.3 Praktinis pritaikymas

Būtina pažymėti, kad GRUB paleidimo posistemė automatiškai įdiegia GDL, tačiau mėginant ją keisti būtų generuojama klaida „triple fault“. Tai reikštų, kad įvyktų kompiuterio perkrovimas. Taigi, pirmiausia reikia apsirašyti naują GDL, vėliau nurodyti procesoriui, kur ji yra. Galiausiai reikia užkrauti naujas reikšmes į šiuos procesoriaus registrus CS, DS, ES, FS ir GS. CS (angl. code segment) registras nurodo, kurį poslinkį naudoti dabartinio vykdomo kodo privilegijų radimui GDL. DS (angl. data segment) analogiškai, tačiau ne kodui, bet duomenims. ES, FS ir GS registrai yra DS alternatyvos ir pavyzdinei programai neturi įtakos.

GDL 64 bitų ilgio reikšmių yra sąrašas. Šios reikšmės apibrėžia, kur prasideda atminties segmentas, kur jis baigiasi bei priėjimo prie jo privilegijas.

Įprasta, kad pirmasis GDL įrašas yra 0, kuris dar vadinamas kaip NULL deskriptorius. Joks segmento registras neturėtų būti nustatytas į nulį, nes tai sukeltų bendrosios apsaugos klaidą (angl. general protection fault). Apie šią ir kitas klaidas detaliau aiškinama skyriuje „3.7 Pertraukimų apdorojimo paprogramės“.

Taip pat kiekvienas GDL įrašas apibrėžia, ar dabartinis segmentas vykdomas supervizoriaus (žiedas 0) ar vartotojo (žiedas 3) režimu. Egzistuoja ir daugiau žiedų, tačiau didžioji dalis operacinių sistemų naudoja tik šiuos du. Pagrindinė taisyklė yra tokia, naudotojo programa sukelia klaidą, kai mėgina pasiekti sisteminius ar nulinio žiedo duomenis. Tai leidžia apsaugoti branduolį nuo lūžimo, kurį galėtų sukelti vykdoma programa. Toks režimas dar vadinamas apsaugotuoju (angl. protected mode).

GDL įrašai leidžia procesoriui nustatyti, ar galima vykdyti tam tikras privilegijuotas komandas. Privilegijuota reiškia, kad tokia komanda gali būti vykdoma tik aukštesniame žiedo lygyje. Tokių asemblerio komandų pavyzdys yra „cli“ ir „sti“, kurios įjungia arba išjungia pertraukimus. Tarkime, kad vartotojo programa gali vykdyti tokias komandas, tuomet ji galėtų sustabdyti sėkmingą branduolio funkcijų vykdymą.

Pirmieji 48 GDL įrašo bitai naudojami segmento adreso ir limito (nuo 0 iki 15 bito) nurodymui. Likę 16 bitai sudaro vadinamuosius priėjimo (angl. access) ir granuliavimo (angl. granularity) laukus:

4 lentelė – GDL įrašo priėjimo laukas

7	6 5	4	3 0
N	Žiedas	DT	Tipas

N – ar segmentas naudojamas;

Ž – žiedo lygis (nuo 0 iki 3);

DT – deskriptoriaus tipas;

Tipas – segmento tipas (kodo/duomenų segmentas).

5 lentelė – GDL įrašo granuliavimo laukas

7	6	5	4	3 0
G	D	0	P	Segmento ilgis (19-16)

G – granuliavimas (0 = 1 baitas, 1 = 4 kilobaitai);

D – operando dydis (0 = 16 bitų, 1 = 32 bitai);

0 – visada 0;

P – ar prieinamas sistemai (beveik visada 0);

Segmento ilgis – nuo 16 iki 19 bito, skirti nurodyti segmento ilgį.

3.5.3.1 GDL įrašų kūrimas

Pavyzdinė programa sukuria tik tris įrašus. Taip yra todėl, kad pirmas įrašas turi būti NULL, skirtas procesoriaus atminties apsaugos galimybėms realizuoti. Kiti du įrašai reikalingi kodo ir duomenų segmentams.

Norint nurodyti procesoriui, kur yra naujoji GDL, reikia naudoti „lgdt“ operaciją. Šiai operacijai reikia paduoti adresą į specialią 48 bitų struktūrą. Pirmieji 16 bitų yra GDL limitas (paskutinis adresas esantis lentelėje). Kiti 32 bitai skirti nurodyti pirmojo GDL įrašo adresui. Šios struktūros apibrėžimo kodas:

```
struct gdt_ptr
{
    unsigned short limit;
    unsigned int base;
} __attribute__((packed));

struct gdt_ptr gp;
```

5 C kodo ištrauka– [gdt.c]

„**__attribute__**((packed))“ reiškia, jog kompiliatoriui draudžiama keisti struktūros išdėstymą, siekiant optimizuoti kodą.

GDL apibrėžimui galime naudoti paprasčiausią masyvą iš trijų elementų:

```
struct gdt_entry
{
    unsigned short limit_low;
    unsigned short base_low;
    unsigned char base_middle;
    unsigned char access;
    unsigned char granularity;
    unsigned char base_high;
} __attribute__((packed));

struct gdt_entry gdt[3];
```

6 C kodo ištrauka – [gdt.c]

Šie apibrėžimai panaudoti faile **[gdt.c]**. Taip pat jame yra nuoroda į funkciją „gdt_flush“. Ši funkciją nurodo procesoriui, kur yra naujasis GDL, ir iš naujo užkrauna segmentų registrus. Paskutinis jos žingsnis yra tolimas šuolis (angl. far jump), perkraunantis kodo segmentą. Šios funkcijos kodas:

```
global gdt_flush   ; leidžia C kodui naudotis šia funkcija
```

```
    extern gp           ; nurodo, kad gp kintamasis apibrėžtas kitame faile
    gdt_flush:
        lgdt [gp]       ; užkrauna GDL naudodamasis mūsų specialia nuoroda
        mov ax, 0x10    ; 0x10 yra duomenų segmento poslinkis naujoje GDL
        mov ds, ax
        mov es, ax
        mov fs, ax
        mov gs, ax
        mov ss, ax
        jmp 0x08:flush2 ; 0x08 yra kodo segmento poslinkis
    flush2:
        ret             ; grįžta į C kodą
```

1 ASM kodo ištrauka – [start.asm]

Taigi, dabar yra apibrėžtos visos funkcijos, kurių reikia GDL užkrovimui ir atminties rezervavimui, tačiau dar liko paskutinė užduotis – GDL įrašų reikšmių įrašymas. Tam atlikti galima pasitelkti žemiau esančias funkcijas. Komentarai puikiai atspindi kodo esmę.

```
    /* Funkcija, kuri apibrėžta faile start.asm,
    * kurios paskirtis yra, tinkamai užkrauti naują GDL */
    extern void gdt_flush();

    /* Nustato deskriptorių GDL viduje */
    void gdt_set_gate(int num, unsigned long base, unsigned long limit, unsigned
char access, unsigned char gran)
    {
        /* Nustato deskriptoriaus bazinį adresą */
        gdt[num].base_low = (base & 0xFFFF);
        gdt[num].base_middle = (base >> 16) & 0xFF;
        gdt[num].base_high = (base >> 24) & 0xFF;

        /* Nustato deskriptoriaus limitus */
        gdt[num].limit_low = (limit & 0xFFFF);
        gdt[num].granularity = ((limit >> 16) & 0x0F);

        /* Nustato granuliškumą ir priėjimo teises */
        gdt[num].granularity |= (gran & 0xF0);
        gdt[num].access = access;
    }
```

7 C kodo ištrauka – [gdt.c]

Funkcijos „gdt_install“ paskirtis yra sukurti naują GDL. Ši funkcija sukuria specialios struktūros nuorodą, kurios pagalba galima užkrauti naująją GDL. Taip pat ji įrašo tris deskriptorius. Pačioje pabaigoje iškviečia gdt_flush(), kuris nurodo procesoriui, kur yra naujoji GDL bei perstato segmentų registrus.

```
void gdt_install()
{
    /* Nustato GDL lentelės nuorodą ir limitą*/
    gp.limit = (sizeof(struct gdt_entry) * 3) - 1;
    gp.base = &gdt;
    /* NULL deskriptorius */
    gdt_set_gate(0, 0, 0, 0, 0);
    /* Antrasis įrašas yra kodo segmentas.
       * Bazinis adresas 0, limitas 4 gigabaitai, granuliškumas 4 kilobaitai,
       *  naudoja 32 bitų  operacijas */
    gdt_set_gate(1, 0, 0xFFFFFFFF, 0x9A, 0xCF);
    /* Lygiai tas pats, kas ankstesnis įrašas, tačiau deskriptoriaus tipas
       * nurodo, kad tai duomenų segmentas */
    gdt_set_gate(2, 0, 0xFFFFFFFF, 0x92, 0xCF);
    /* Senąjį GDL pakeičia naujuoju! */
    gdt_flush();
}
```

8 C kodo ištrauka – [gdt.c]

Norint užkrauti naująją GDL, tereikia faile **[main.c]** iškviesti funkcija „gdt_install“. Deja, šis pakeitimas jokių vizualių pakeitimų demonstracinei programai nesuteikia, nes tai vidinis sistemos veikimo pakeitimas.

3.6 Pertraukimų deskriptorių lentelė

3.6.1 Apibrėžimas

Pertraukimų deskriptorių lentelė (PDL) naudojama nurodyti procesoriui, kokią pertraukimų apdorojimo paprogramę (PAP) iškviesti klaidos ar asemblerio „int“ operacijos atveju. PDL įrašai dar vadinami pertraukimų užklausomis tuo atveju, jei įrenginys baigė vykdyti užklausą ir turi būti aptarnautas. Pertraukimų apdorojimo paprogramių ir klaidų veikimas detaliau paaiškintas skyriuje „3.7 Pertraukimų apdorojimo paprogramės“.

PDL įrašai yra panašūs į GDL. Abu turi bazinį adresą, priėjimo vėliavas (angl. flag) bei yra 64 bitų ilgio. Tačiau iš tiesų skiriasi šių laukų reikšmės. PDL deskriptoriuje nurodytas bazinis adresas yra PAP adresas, kurią procesorius turi iškviesti, kai įvyksta pertraukimas. PDL įrašas

neturi limito (angl. limit), vietoj jo reikia nurodyti segmentą, kuriame yra PAP. Toks mechanizmas įgalina procesorių valdymą perduoti branduoliui, kai įvyksta pertraukimas skirtinguose žieduose (pavyzdžiui, kai yra vykdoma naudotojo programa).

PDL įrašo priėjimo laukas yra panašus į GDL įrašo: yra laukelis, kuris nurodo ar deskriptorius naudojamas, tuomet nurodomas deskriptoriaus privilegijų lygis, skirtas nurodyti aukščiausią žiedo numerį, kuriam leidžiama naudoti šį pertraukimą. Paskutinieji 5 bitai visuomet nustatomi į 01110b (14). Grafiškai šis priėjimo baitas atrodo taip:

6 lentelė – pertraukimų deskriptorių lentelės įrašo priėjimo baitas

7	6 5	4 3 0
N	Žiedas	Visada 01110b

N – ar segmentas naudojamas;

Ž – žiedo lygis (nuo 0 iki 3).

3.6.2 Praktinis panaudojimas

Faile [**idt.c**] yra apibrėžtos struktūros PDL įrašams ir specialiai nuorodai, kuri kaip ir GDL atveju yra reikalinga tam, kad būtų galima sėkmingai užkrauti PDL. Taip pat šiame faile yra masyvas, susidedantis iš 256 PDL įrašų, kuris yra būsima PDL. Programiniame kode, tai atrodo štai taip:

```
/* Apibrėžia PDL įrašą */
struct idt_entry
{
    unsigned short base_lo;
    unsigned short sel;        /* Branduolio segmentas */
    unsigned char always0;     /* Visada nulis */
    unsigned char flags;
    unsigned short base_hi;
} __attribute__((packed));
struct idt_ptr
{
    unsigned short limit;
    unsigned int base;
} __attribute__((packed));
/* 256 PDL įrašų deklaravimas. Tačiau pavyzdinė programa naudos
 * tik 32. Jei įvyksta neapibrėžtas pertraukimas, tuomet tai sukelia
 * „Unhandled Interrupt“ klaidą. Be to, kiekvienas deskriptorius,
 * kurio naudojamumo bitas 0, generuoja tokią pat klaidą */
struct idt_entry idt[256];
```

```
struct idt_ptr idtp;
```

9 C kodo ištrauka – [idt.c]

Norint užkrauti PDL reikia pasinaudoti operacija „lidt“. Užkrovimą atliekanti funkcija esanti faile [**start.asm**]:

```
; Užkrauna PDL, pasinaudodama specialia nuoroda idtp.
; Ši funkcija kviečiama iš C kodo
global idt_load
extern idtp
idt_load:
    lidt [idtp]
    ret
```

2 ASM kodo ištrauka – [start.asm]

Turint struktūras, PDL įrašų masyvą ir užkrovimo funkciją, telieka aprašyti įrašų sukūrimo ir PDL iniciavimo funkcijas:

```
/* Funkcija skirta PDL įrašų nustatymui */
void idt_set_gate(unsigned char num, unsigned long base, unsigned short sel,
unsigned char flags)
{
    /* Pertraukimo popragramės bazinis adresas */
    idt[num].base_lo = (base & 0xFFFF);
    idt[num].base_hi = (base >> 16) & 0xFFFF;

       /* Segmentas arba selektorius (angl. selector), kurį
       * naudos šis PDL įrašas */
    idt[num].sel = sel;
    idt[num].always0 = 0;
    idt[num].flags = flags;
}
```

10 C kodo ištrauka – [idt.c]

Taip pat reikia aprašyti „idt_install“ funkciją. Jos paskirtis sukurti specialią PDL nuorodą, išvalyti deskriptorių lentelę ir perduoti jos adresą procesoriui. Ši funkcija yra analogiška „gdt_install“ (8 C kodo ištrauka – [gdt.c]) funkcijai. Visą PDL užkrovimo funkcijos realizacija pateikta priede (23 C kodo ištrauka – [idt.c]).

Taigi, norint įdiegti PDL tereikia iškviesti funkciją „idt_install“. Įdiegus PDL ir įvykdžius neleistiną operaciją, pavyzdžiui, dalybą iš nulio, kompiuteris persikrautų. Norint valdyti

įvykstančius pertraukimus, reikia į sistemą įdiegti pertraukimų apdorojimo paprogrames. Plačiau apie tai kitame skyriuje.

3.7 Pertraukimų apdorojimo paprogramės

3.7.1 Apibrėžimas

Pertraukimų apdorojimo paprogramės (PAP) yra skirtos išsaugoti procesoriaus būseną ir tinkamai nustatyti procesoriaus registrus, prieš iškviečiant branduolyje C kalba parašytą tam skirtą funkciją. Visiems šiems veiksmams atlikti pakanka apie 20 asemblerio kodo eilučių. Svarbu, kad PDL įrašas rodytų į jam skirtą PAP, kad įvykęs pertraukimas būtų sėkmingai apdorotas.

Klaida yra toks įvykis, kuomet procesorius susiduria su situacija, kurios negali toliau vykdyti, pavyzdžiui, dalyba iš nulio. Tokiu atveju yra generuojama klaida, kad branduolys galėtų sustabdyti tai vykdantį procesą ar užduotį bei išvengti sistemos lūžimo.

Pirmieji 32 PDL įrašai atsakingi už klaidas, kurias generuoja pats procesorius, todėl privalo būti apdorojami. Taip pat, kai kurios klaidos į steką įdeda papildomą reikšmę – klaidos kodą, kurio reikšmė priklauso nuo jį įdėjusios klaidos specifikos. Klaidų sąrašas:

7 lentelė – generuojamų klaidų (pertraukimų) sąrašas

Nr.	Pavadinimas	Ar turi klaidos kodą?
0	Dalyba iš nulio (division by zero exception)	Ne
1	Derinimo klaida (debug exception)	Ne
2	Nemaskuojamo pertraukimo klaida (non maskable interrupt exception)	Ne
3	Kontrolinio stabdymo klaida (breakpoint exception)	Ne
4	Sveikųjų rėžių peržengimo klaida (into detected overflow exception)	Ne
5	Pateikimo už ribų (out of bounds exception)	Ne
6	Klaidingas operacijos kodas (invalid opcode exception)	Ne
7	Koprocesoriaus nepasirengimo klaida (no coprocessor exception)	Ne
8	Dvigubo sutrikimo klaida (double fault exception)	Taip
9	Koprocesoriaus segmento peržengimo klaida (coprocessor segment overrun exception)	Ne
10	Blogo užduočių būsenų segmento klaida (bad tss exception)	Taip
11	Segmento nenaudojamumo klaida (segment not present exception)	Taip
12	Steko sutrikimo klaida(stack fault exception)	Taip
13	Bendrosios apsaugos klaida(general protection fault exception)	Taip

14	Puslapiavimo klaida (page fault exception)	Taip
15	Nežinomo pertraukimo klaida (unknown interrupt exception)	Ne
16	Koprocesoriaus klaida (coprocessor fault exception)	Ne
17	Išlygiavimo tikrinimo klaida (alignment check exception (486+))	Ne
18	Mašinos tikrinimo klaida (machine check exception (Pentium/586+))	Ne
19-31	Rezervuotos klaidos (reserved exceptions)	Ne

Kaip matyti iš 7 lentelės, kai kurios klaidos patalpina klaidos kodą steke. Siekiant supaprastinti situaciją, galima visais atvejais siųsti klaidos kodą į steką. Jei kodas nėra siunčiamas, tuomet į steką įdedamas nulis. Tokiu būdu paliekama mažiau vietos klaidoms susijusioms su steku. Taip pat reikia nepamiršti, kad siekiant galimybės žinoti, kuri klaida įvyko, reikia į steką patalpinti pertraukimo numerį.

3.7.2 Praktinis panaudojimas

Kadangi siekiama, kad pateikiami pavyzdžiai būtų paprasti ir aiškūs, kiekviena pertraukimų apdorojimo paprogramių žymė (angl. label) nušoka į „isr_common_stub“. Taip pat kiekviena PAP žymė kviečia asemblerio funkciją „cli“, kuri išjungia pertraukimus ir neleidžia įvykti PU (angl. IRQ). Jei taip įvyktų, tikėtina, kad branduolys nulūžtų. Kode tai atrodo taip:

```
; Pertraukimų apdorojimo žymės
global isr0
global isr1
...                ; įrašyti trūkstamus
global isr31
; 0: Dalybos iš nulio (devide by zero exception)
isr0:
    cli
    push byte 0    ; patalpinam fiktyvų klaidos kodą, kad išlaikyti vientisumą
    push byte 0    ; PAP numeris, pvz., isr2 atveju reikia push byte 2
    jmp isr_common_stub
...                            ; įrašyti trūkstamus
; 8: Dvigubo sutrikimo klaida (su klaidos kodu) (double fault exception)
isr8:
    cli
    push byte 8                ; į steką netalpinamas  fiktyvus kodas,
    jmp isr_common_stub        ; nes jis bus automatiškai patalpinas
...                            ; įrašyti trūkstamus
; 31: Rezervuota klaida
isr31:
    cli
```

```
        push byte 0
        push byte 31
        jmp isr_common_stub
```

3 ASM kodo ištrauka – [start.asm]

Žymė „isr_common_stub“ išsaugo procesoriaus būseną steke, patalpina dabartinio steko adresą į steką (reikalinga, kad C kalba parašyta funkcija galėtų jį pasiekt) bei iškviečia funkciją „fault_handler“. Po funkcijos iškvietimo, atstato būseną pasinaudodamas steku.

```
    ; C faile esanti funkcija
    extern fault_handler
    ;  Išsaugo procesoriaus būseną, iškviečia C kalbos funkciją,
    ;  galiausiai atstato būseną iš steko
    isr_common_stub:
        pusha
        push ds
        push es
        push fs
        push gs
        mov ax, 0x10   ; užkrauna branduolio duomenų segmento deskriptorių
        mov ds, ax
        mov es, ax
        mov fs, ax
        mov gs, ax
        mov eax, esp
        push eax
        mov eax, fault_handler   ; C kalboje parašyta funkcija
        call eax
        pop eax
        pop gs
        pop fs
        pop es
        pop ds
        popa
        add esp, 8               ; Išvalo klaidos kodą ir PAP numerį
        sti
        iret                     ; iš steko paima penkias reikšmes:
                                 ; CS, EIP, EFLAGS, SS ir ESP
```

4 ASM kodo ištrauka – [start.asm]

C kalboje (**[isrs.c]**) reikia apibrėžti nuorodas į PAP, aprašytas **[start.asm]** faile. To reikia tam, kad būtų galima susieti PAP su atitinkamomis PDL reikšmėmis.

```
/* Funkcijų prototipai visoms doroklėms:
* pirmieji 32 PDL įrašai yra rezervuoti ir turi būti aptarnaujami */
extern void isr0();
extern void isr1();
...               ; įrašyti trūkstamus
extern void isr30();
extern void isr31();
/* Nėra būdo gauti visų funkcijų vardus, todėl funkcija vykdo
* daug beveik identiškų sakinių.
* Pirmieji 32 PDL lentelės įrašai rodo į
* pirmuosius 32 PAP.  Priėjimo vėliava nustatoma į 0x8E.
* Tai reiškia, kad įrašas naudojamas, vykdomas žiede 0 bei
* jo paskutiniai 5 bitai nustatyti į reikiamą skaičių – 14,
* šešioliktainiame pavidale E. */
void isrs_install()
{
    idt_set_gate(0, (unsigned)isr0, 0x08, 0x8E);
    idt_set_gate(1, (unsigned)isr1, 0x08, 0x8E);
    idt_set_gate(2, (unsigned)isr2, 0x08, 0x8E);
       ...         ; įrašyti trūkstamus
    idt_set_gate(30, (unsigned)isr30, 0x08, 0x8E);
    idt_set_gate(31, (unsigned)isr31, 0x08, 0x8E);
}
```

11 C kodo ištrauka – [isrs.c]

Paprasčiausias būdas apdoroti pertraukimus yra aprašyti funkciją, kuri išveda įvykusio pertraukimo pavadinimą ir vykdo begalinį ciklą, taip sustabdydama sistemos darbą. Pertraukimo pavadinimai saugomi tam skirtame masyve „exception_messages“.

```
/* Paprastas eilučių masyvas, saugantis kiekvienos klaidos pavadinimą,
*  kuris išgaunamas taip: exception_message[pertraukimo_numeris] */
unsigned char *exception_messages[] =
{
    „Division By Zero“,
    „Debug“,
    „Non Maskable Interrupt“,
       ...         ; įrašyti trūkstamus
    „Reserved“,
    „Reserved“
```

```
};
/* Visos PAP rodo į šią funkciją, kuri tiesiog pasako,
* kokia klaida įvyko ir sustabdo sistemą, pasinaudodama begaliniu ciklu.
* Kiekviena PAP išjungia pertraukimų
* mechanizmą, kad būtų išvengta PRK (IRQ) įvykimo, kuris
* galėtų sugadinti branduolio duomenų struktūras */
void fault_handler(struct regs *r)
{
    if (r->int_no < 32)
    {
        puts(exception_messages[r->int_no]);
        puts(„ Klaida. Systema sustabdyta!\n“);
        for (;;);
    }
}
```

12 C kodo ištrauka – [isrs.c]

Aukščiau esančiame kode naudojama nauja struktūra „regs“. Ji naudojama C kalba parašytam kodui parodyti, kaip atrodo stekas. Būtina atsiminti, kad faile [**start.asm**] į steką yra patalpinama nuoroda į patį steką tam, kad būtų galima gauti klaidos kodą ir pertraukimo numerį. Toks dizainas leidžia kiekvienai skirtingai PAP naudoti ta pačią funkciją, kuri pasako, kokia klaida įvyko. Struktūros apibrėžimas:

```
/* Ši struktūra apibrėžia steką po kurios nors PAP žymės įvykdymo */
struct regs
{
   unsigned int gs, fs, es, ds;           // segmentų adresai
       // patalpino operacija „pusha“
   unsigned int edi, esi, ebp, esp, ebx, edx, ecx, eax;
       // pertraukimo numeris ir klaidos kodas
   unsigned int int_no, err_code;
       // automatiškai procesoriaus patalpintos reikšmės
   unsigned int eip, cs, eflags, useresp, ss;
};
```

13 C kodo ištrauka – [system.h]

Norint išbandyti, kaip veikia pertraukimų apdorojimas galima sukelti kokią nors klaidą, pavyzdžiui, dalybą iš nulio. Tam galima pasinaudoti šiuo kodu „putch(1 / 0)“.

3.8 Pertraukimų užklausos ir programuojami pertraukimų valdikliai

3.8.1 Apibrėžimas

Pertraukimų užklausos (PU) yra pertraukimai, kuriuos sukelia aparatūrinė įranga. Pavyzdžiui, įrenginys turi paruoštų duomenų, kuriuos reikia nuskaityti, arba įrenginys baigė rašyti buferį į diską. Apibendrintai galima teigti, kad įrenginys generuoja PU tada, kai jam reikalingas procesoriaus dėmesys. PU generuoja visi įrenginiai pradedant tinklo ir garso plokštėmis ir baigiant pele, klaviatūra ir nuosekliąja jungtimi (angl. serial port).

Bet koks kompiuteris, suderinamas su 286 ir naujesniu procesoriumi (IBM PC/AT), turi du lustus (angl. chip), kurie naudojami PU valdymui. Šie du lustai vadinami programuojamais pertraukimų valdikliais (PPV). Jie turi dvi roles: vienas yra valdantysis (angl. master), kitas – pagalbinis (angl. slave). Valdantysis valdiklis prijungtas tiesiai prie procesoriaus, kad galėtų siųsti signalus. Tuo tarpu pagalbinis prijungtas prie valdančiojo valdiklio per PU2.

Kiekvienas PPV gali apdoroti 8 PU: valdantysis valdiklis nuo 0 iki 7, pagalbinis - nuo 8 iki 15. Kadangi antrasis valdiklis prijungtas prie pagrindinio per PU2, kaskart įvykus PU nuo 8 iki 15, iškart įvyksta PU2.

Kai įrenginys išsiunčia signalą su PU, įvyksta pertraukimas ir procesorius nustoja vykdyti operacijas, kad iškviestų atitinkamą PAP. Tuomet procesorius atliekama reikiamus veiksmus, pavyzdžiui, nuskaito duomenis iš klaviatūros), po to privalo informuoti PPV, kad sėkmingai įvykdė paprogramę. Procesorius atsakymą PPV pateikia įrašydamas komandos baitą 0x20 į atitinkamo PPV komandų registrą. Valdančiojo valdiklio komandos registras yra įvedimo/išvedimo prievade adresu 0x20, tuo tarpu pagalbinio valdiklio – adresu 0xA0.

Taip pat reikia įsidėmėti, kad standartiškai PU nuo 0 iki 7 rodo į PDL įrašus nuo 8 iki 15. Tuo tarpu PU nuo 8 iki 15 surišti su PDL įrašais nuo 0x70 iki 0x78. Tačiau PDL įrašai nuo 0 iki 31 yra rezervuoti klaidoms, kurias signalizuoja procesorius, todėl reikia pakeisti pertraukimų numerius. Po šio veiksmo PU nuo 0 iki 15 rodys į PAP nuo 32 iki 47.

3.8.2 Praktinis pritaikymas

Žemiau pateikti veiksmai labai panašūs į PAP programavimą. Skiriasi tik pertraukimų numeriai, vietoj isr_common_stub naudojama žymė irq_common_stub bei jame kviečiama kita C kalbos funkcija („irq_handler“). [**start.asm**] faile esantis kodas, atsakingas už PU apdorojimą:

```
    global irq0
    ...    ; įrašyti trūkstamus
    global irq15
    ; 32: PU0 (IRQ0)
    irq0:
```

```
        cli
        push byte 0    ; kaip ir PAP atveju, patalpinamas netikras klaidos
        push byte 32   ; kodas tam, kad būtų išlaikytas vientisumas
        jmp irq_common_stub
...     ; įrašyti trūkstamus
; 47: PU15 (IRQ15)
irq15:
        cli
        push byte 0
        push byte 47
        jmp irq_common_stub
extern irq_handler
; Iškviečia funkciją esančią faile „irc.c“
irq_common_stub:
; Įterpti reikalingą kodą
```

5 ASM kodo ištrauka – [start.asm]

Žymės „irq_common_stub“ kodas analogiškas „isr_common_stub“ žymei, skiriasi tik jame iškviečiama C funkcija: vietoje „fault_handler“ naudojama „irq_handler“. Šios žymės realizacija pateikta priede (6 ASM kodo ištrauka – [start.asm]).

Standartiškai, PU nuo 0 iki 7 rodo į PDL įrašus nuo 8 iki 15. Tai yra problema, nes šios reikšmės yra jau užimtos, todėl PU nuo 0 iki 15 yra perprogramuojamos į PDL įrašus nuo 32 iki 47.

```
void irq_remap(void)
{
    outportb(0x20, 0x11); // pasiunčiama iniciavimo komanda
    outportb(0xA0, 0x11); // pasiunčiama iniciavimo komanda
    outportb(0x21, 0x20); // pagrindinis valdiklis - nuo 32
    outportb(0xA1, 0x28); // pagalbinis valdiklis - nuo 40
    outportb(0x21, 0x04);
    outportb(0xA1, 0x02);
    outportb(0x21, 0x01);
    outportb(0xA1, 0x01);
    outportb(0x21, 0x0);
    outportb(0xA1, 0x0);
}
```

14 C kodo ištrauka – [irq.c]

Liko suprogramuoti kelias funkcijas C kalba:

```
/* Tai yra PAP, kurie rodo į specialų PU dorotoją vietoj to,
```

```
    * kad rodytų į tipinę funkiją „fault_handler“, skirtą kitoms PAP */
    extern void irq0();
    ... // įrašyti trūkstamus
    extern void irq15();
    /* Funkcijų nuorodų masyvas, kurio paskirtis saugoti
    * funkcijų adresus. Vėliau še adresai naudojami
    * PU apdorojimui */
    void *irq_routines[16] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
    /* Įdiegia nestandartinę PU doroklę pateiktam PU pagal numerį*/
    void irq_install_handler(int irq, void (*handler)(struct regs *r))
    {
        irq_routines[irq] = handler;
    }
    /* Pirma perprogramuojami pertraukimų valdikliai (irq_remap).
    *  Po to PAP nustatomos į atitinkamus PDL įrašus. */
    void irq_install()
    {
        irq_remap();
        idt_set_gate(32, (unsigned)irq0, 0x08, 0x8E);
        ... // įrašyti trūkstamus
        idt_set_gate(47, (unsigned)irq15, 0x08, 0x8E);
    }
```

15 C kodo ištrauka – [irq.c]

Kiekviena PU paprogramė iškviečia „irq_handler“ funkciją, kuri įvykdo konkrečiam PU skirtą doroklę (angl. handler) ir PPV praneša, kad jie buvo aptarnauti, t.y., nusiunčia jiems komandą – „pertraukimo pabaiga“ (PP).

Yra du PPV: pirmas įvedimo/išvedimo prievade adresu 0x20, antras – 0xA0. Jei įvyksta pertraukimas pagalbiniame valdiklyje (PU nuo 8 iki 15), tuomet apie apdorojimo pabaigą reikia informuoti abu valdiklius. Kitu atveju reikia tik nusiųsti PP į pirmąjį valdiklį. Jei šis pranešimas nebus nusiųstas, tai neįvyks nė vienas naujas PU.

```
    void irq_handler(struct regs *r)
    {
        /* Tuščia funkcijos nuoroda */
        void (*handler)(struct regs *r);
        /* Išsiaiškina ar yra paruošta konkrečiam PU skirta doroklė */
        handler = irq_routines[r->int_no - 32];
        if (handler)
            handler(r);
        /* Jei įvykio aukštesnio numerio pertraukimas nei 40
```

```
            * (atitinkamai PU 8 – 15),  tada reikia nusiųsti PP į antrąjį
            * valdiklį */
          if (r->int_no >= 40)
              outportb(0xA0, 0x20);
          /* Bet kuriuo atveju PP reikia siųsti į pagrindinį valdiklį */
          outportb(0x20, 0x20);
      }
```

16 C kodo ištrauka – [irq.c]

Tam, kad PU valdančios PAP būtų įdiegtos, tereikia iškviesti funkciją „irq_install“. Tai galima padaryti failo [**main.c**] funkcijoje „main“. Po PAP įdiegimo reikia įvykdyti komandą „__asm__ __volatile__ („sti“);“, kuri įjungia PU vyksmą.

Sekančiame skyriuje galima pamatyti, kaip panaudojama „irq_install_handler“ funkcija, siekiant įdiegti specialią paprogramę programuojamo intervalo taimeriui.

3.9 Programuojamo intervalo taimeris – sistemos laikrodis

3.9.1 Apibrėžimas

Programuojamo intervalo taimeris (PIT), taip pat vadinamas sistemos laikrodžiu, yra labai vertingas lustas galintis tiksliai generuoti pertraukimus nustatytais laiko intervalais.

Lustas turi 3 kanalus:

0. kanalas yra surištas su PU0 (skirtas pertraukti procesorių nustatytu reguliariu laiku). Šis vienas taimerio kanalas leidžia tiksliai suplanuoti naujus procesus, bei leidžia vykdomam procesui palaukti nustatytą laiko periodą. Pagal nutylėjimą, šis kanalas yra nustatytas sukelti PU0 18,222 karto per sekundę. Toks lusto pertraukimų generuojamas dažnis buvo nustatytas tam, kad vienas taktas būtų lygiai 0,055 sekundės. Naudojant 16 bitų taktų skaitliuką, skaitliukas pasieks maksimalią reikšmę ir persivers į 0 kas valandą.
1. kanalas senesniuose kompiuteriuose buvo naudojamas (kartu su tiesioginės kreipties atminties valdikliu) norint atnaujinti operatyviosios atminties duomenis. Dažniausiai kiekvienas operatyviosios atminties bitas sudarytas iš kondensatoriaus, kuris laiko nedidelį krūvį, nusakantį to bito būseną. Dėl krūvio nutekėjimo, kondensatoriai turi būti iš naujo įkraunami, kad jie „nepamirštų“ savo būsenos. Naujuose kompiuteriuose operatyviosios atminties atnaujinimas vyksta pasitelkus tam skirtą komplikuotą aparatūrą ir šis lustas nebėra naudojamas.
2. kanalas surištas su sistemos garsiakalbiu taip, kad kanalu nurodytas dažnis nustato garso dažnį.

Pasinaudojus „outportb“ funkcija įrašyti duomenis į įvesties/išvesties prievadą, galima nustatyti dažnį, kuriuo taimeris sukelia PU0. Kiekvienam iš 3 PIT kanalui yra atskiras duomenų registras įvedimo/išvedimo prievado adresuose 0x40, 0x41 ir 0x42. Adrese 0x43 yra valdymo registras. Siunčiami duomenys iš tikrųjų yra daliklis, o ne taktų dažnis. Taimeris dalina savo dažnį, kuris yra 1,19 MHz (1193180Hz), iš skaičiaus nurodyto duomenų registre ir taip gauna, kiek kartų per sekundę reikia sukelti signalą tam kanalui. Iš pradžių reikia pasirinkti kanalą, kuris bus keičiamas, pasinaudojant valdymo registru prieš rašant į duomenų registrą.

8 lentelė – valdymo registro struktūra

7 6	5 4	3 1	0
K	RS	VR	KR

- K – kanalo nr.:
  - 00 – 0 kanalas;
  - 01 – 1 kanalas;
  - 10 – 2 kanalas;
- RS – rašymo/skaitymo režimas:
  - 01 – jaunesnysis baitas;
  - 10 – vyresnysis baitas;
  - 11 – vyresnysis, po to jaunesnysis baitas.
- VR – veikimo režimas:
  - 000 – 0 režimas (Interrupt on terminal count);
  - 001 – 1 režimas (Hardware Retriggerable one shot);
  - 010 – 2 režimas (Rate Generator);
  - 011 – 3 režimas (Square Wave Mode);
  - 100 – 4 režimas (Software Strobe);
  - 101 – 5 režimas (Hardware Strobe);
- KR – kanalo režimas:
  - 0 – 16 bitų skaičius dvejetainėje sistemoje;
  - 1 – 4 dešimtainiai skaitmenys.

3.9.2 Praktinis panaudojimas

Pasirenkamas 0 kanalas. Daliklio reikšmė, kurią reikia įrašyti į duomenų registrą, yra 16 bitų dvejetainė reikšmė, todėl reikia nustatyti atitinkamą kanalo režimą ir perkelti abu baitus. Taip pat tarkime norima nustatyti 3 veikimo režimą (Square Wave). Gauta reikšmė, kurią reikia nusiųsti į valdymo registrą, yra 0x36 (K – 0, RS – 11, VR – 011, KR – 0; gaunama 0110110b) . Tai galima pritaikyti toliau pateiktoje funkcijoje.

Pastabos:

- *Patartina realiame branduolyje nustatyti 100Hz dažnį.*
- *Plačiau pasiskaityti apie PIT veikimo režimas galima internete adresais:*
  - *< http://www.qzx.com/pc-gpe/pit.txt >;*
  - *< http://wiki.osdev.org/Programmable_Interval_Timer >.*

```
Void timer_phase(int hz)
{
    int divisor = 1193180 / hz;       /* Suskaičiuojame daliklį */
    outportb(0x43, 0x36);     /* Nustatome valdymo registro reikšmę į 0x36 */
    outportb(0x40, divisor & 0xFF);   /* Nustatome jaunesnyjį daliklio baitą */
    outportb(0x40, divisor >> 8);   /* Nustatome vyresnyjį daliklio baitą */
}
```

17 C kodo ištrauka – [timer.c]

Sukuriamas failas **[timer.c]**. Toliau pateiktame kode galima pastebėti, kad yra skaičiuojama kiek taktų praėjo. Tai gali būti panaudota kaip sistemos veikimo laiko skaitiklis kai branduolys taps sudėtingesnis. Taimeris pagal nutylėjimą naudoja 18,222HZ dažnį, kad žinotų, kada pateikti užrašą „Praėjo viena sekunde“ kas sekundę. Laikrodžio dažnį galima nustatyti bet kur kode, tačiau patartina tai padaryti „timer_install“ funkcijoje ir išlaikyti tvarką.

```
/* Saugo kiek taktų sistema veikia */
int timer_ticks = 0;
/* Apdoroja PIT. Šiuo konkrečiu atveju labai paprastai:
*  kiekvieną kartą įvykus taimerio pertraukimui „timer_ticks“ padidanamas
*  vienetu. Pagal nutylėjimą pertraukimai įvyksta 18,222 karto per sekundę. */
void timer_handler(struct regs *r)
{
    /* Taktų skaitliukas padidinamas vienetu */
    timer_ticks++;
    /* Kas 18 taktų (apytiksliai kas sekundę) išvedame pranešimą */
    if (timer_ticks % 18 == 0)
        puts(„Praejo viena sekunde\n“);
}
/* Sukonfigūruoja taimerį */
void timer_install()
{
    /* Įdiegia „timer_handler“ į PU0 */
    irq_install_handler(0, timer_handler);
```

```
    }
```

18 C kodo ištrauka – [timer.c]

Primenama, kad reikia iškviesti funkciją „timer_install“ iš „main“ funkcijos **[main.c]** faile. Taip pat reikia nepamiršti įdėti funkcijos aprašą į **[system.h]**. Toliau pateiktas kodas pademonstruoja taimerio mechanizmo panaudojimą. Ši funkcija palaukia, kol duotas laikas „ticks“ praeina.

```
/* Ciklas suksis, kol bus pasiekta duoto laiko pabaiga */
void timer_wait(int ticks)
{
    unsigned long eticks;

    eticks = timer_ticks + ticks;
    while(timer_ticks < eticks);
}
```

19 C kodo ištrauka – [timer.c]

3.10 Klaviatūros valdymas ir duomenų skaitymas

3.10.1 Aprašymas

Klaviatūra yra vienas labiausiai paplitusių būdų naudotojui įvesti informaciją į kompiuterį, todėl labai svarbu sukurti kažkokią tvarkyklę, kuri valdytų klaviatūrą ir duomenų srautus su ja. Šiame skyriuje pateikta svarbiausia informacija: kaip sužinoti klaviatūros įvesties kodą (angl. scancode) po klavišo paspaudimo ir kaip vėliau paversti gautą kodą į standartinį ASCII simbolį.

Klaviatūros įvesties kodas – tai paprastas skaičius. Klaviatūra priskiria tam tikrą skaičių kiekvienam klavišui. Klaviatūros įvesties kodai yra dažniausiai numeruojami iš viršaus žemyn ir iš kairės į dešinę su keletu išimčių, kurios užtikrina atgalinį suderinamumą su senomis klaviatūromis.

3.10.2 Praktinis panaudojimas

3.10.2.1 Skaitymas iš klaviatūros

Norint klaviatūros įvesties kodus paversti į ASCII reikšmes, reikia apsibrėžti ir realizuoti klavišų išdėstymą. Tai galima padaryti apsibrėžiant masyvą taip, kad būtų galima lengvai pagal klaviatūros įvesties kodą rasti norimą reikšmę. Taip pat reikia pažymėti, kad jei 7 bitas yra nustatytas (tai galima patikrinti atliekant operaciją „scancode & 0x80“), taip yra pranešama, jog klavišas buvo ką tik atleistas.

Toliau pateiktas pavyzdinis klaviatūros simbolių išdėstymas. Tai išsaugota masyve, todėl nesunkiai galima gauti atitinkamą reikšmę. Jis skirtas standartinei angliškai klaviatūrai.

```
unsigned char kbdeng[128] =
{
    0,  27, ‚1‘, ‚2‘, ‚3‘, ‚4‘, ‚5‘, ‚6‘, ‚7‘, ‚8‘,   /* 9 */
  ‚9‘, ‚0‘, ‚-‚, ‚=‘, ‚\b‘,                           /* Backspace */
  ‚\t‘,                                              /* Tab */
  ‚q‘, ‚w‘, ‚e‘, ‚r‘,         /* 19 */
  ‚t‘, ‚y‘, ‚u‘, ‚i‘, ‚o‘, ‚p‘, ‚[‚, ‚]‘, ‚\n‘,       /* Enter */
    0,                                              /* 29   - Control */
  ‚a‘, ‚s‘, ‚d‘, ‚f‘, ‚g‘, ‚h‘, ‚j‘, ‚k‘, ‚l‘, ‚;‘,   /* 39 */
 ‚\‘‘, ‚`‘,   0,                                     /* Kairysis shift */
 ‚\\‘, ‚z‘, ‚x‘, ‚c‘, ‚v‘, ‚b‘, ‚n‘,                  /* 49 */
  ‚m‘, ‚,‘, ‚.‘, ‚/‘,   0,                           /* Dešinysis shift */
  ‚*‘,
    0, /* Alt */ ‚ ‚, /* Space bar */ 0, /* Caps lock */
    0, /* 59 – F1 key ... > */ 0,   0,   0,   0,   0,   0,   0,   0,
    0, /* < ... F10 */ 0, /* 69 – Num lock*/ 0, /* Scroll Lock */
    0, /* Home key */ 0, /* Rodyklė į viršų*/ 0, /* Page Up */
  ‚-‚, 0, /* Rodyklė į kairę */ 0, 0, /* Rodyklė į dešinę */
  ‚+‘, 0, /* 79 – End key*/ 0, /* Rodyklė žemyn */ 0, /* Page Down */
    0, /* Insert Key */ 0, /* Delete Key */ 0,   0,   0, 0, /* F11 Key */
    0, /* F12 Key */ 0, /* Visi kiti klavišai neapibrėžti */
};
```

20 C kodo ištrauka – [kbd.c]

Paversti klaviatūros įvesties kodą į ASCII simbolį galima labai paprastai, tereikia pasinaudoti šiuo kodu:

```
char mychar = kbdeng[scancode];
```

Pastaba. Nors komentaruose ir pažymėti funkciniai bei „alt“, „shift“, „control“ klavišų kodai, jie masyve aprašyti kaip „0“: šiuo atveju reikia sugalvoti atsitiktines reikšmes, kaip kad pavyzdžiui, ASCII paprastai nenaudojamos reikšmės. Įgyvendinimui pasiūlymas: reiktų apsibrėžti globalų kintamąjį, kurį būtų galima naudoti kaip funkcinių klavišų būsenos indikatorių. Šis indikatoriaus kintamasis galėtų turėti vieną bitą nustatytą „alt“ klavišui, vieną – „control“ ir vieną – „shift“. Taip pat gera mintis turėti „capslock“, „numlock“ ir „scrolllock“ klavišams.

Klaviatūra yra prijungta prie kompiuterio per specialų pagrindinės plokštės lustą. Šis lustas turi 2 kanalus: vieną klaviatūrai ir vieną pelei. Klaviatūros valdiklis, kaip sistemos įrangos

komponentas, turi adresą įvesties/išvesties magistralėje, kurį galime panaudoti kreipimuisi ir kontrolei. Klaviatūros valdiklis turi 2 pagrindinius registrus: duomenų registrą 0x60 įvedimo/išvedimo prievado adrese ir valdymo registrą 0x64 adrese. Viskas ką klaviatūra nori perduoti kompiuteriui yra saugoma duomenų registre. Klaviatūra sukelia PU1, kai ji turi duomenų perskaitymui. Toliau pateiktas kodas paverčia klaviatūros įvesties kodą į ASCII reikšmę ir simbolis išvedamas į ekraną. Galima sugalvoti daug skirtingų realizacijų. Pavyzdžiui, galima panaudoti būsenos kintamuosius ir nustatyti ar buvo klavišas paspaustas kartu su „shift“ ir tada panaudoti kitą klavišų išdėstymo lentelę. Arba pridėti papildomai 128 reikšmes kbdeng masyvui (20 C kodo ištrauka – [kbd.c]) ir, jei paspaustas „shift“ klavišas, pridėti prie klaviatūros įvesties kodo 128.

```
/* Apdoroja klaviatūros pertraukimus */
void keyboard_handler(struct regs *r)
{
    unsigned char scancode;
    /* Nuskaitome duomenis iš klaviatūros duomenų registro */
   scancode = inportb(0x60);
    /* Jei vyriausias bitas yra nustatytas, tai reiškia, kad klavišas
    *  buvo ką tik atleistas */
    if (scancode & 0x80)
    {
       /* Čia galima patikrinti ar naudotojas atleido „shift“, „alt“ arba
        *  „control“ klavišus. */
    }
    else
    {
        /* Čia klavišas buvo paspaustas. Reikia atkreipti dėmesį, kad
        *  jei klavišas yra laikomas nuspaustas, tai generuoja
        *  pasikartojančius klavišo paspaudimus. */
        putch(kbdeng[scancode]);
    }
}
```

21 C kodo ištrauka – [kbd.c]

Matyti, kad klaviatūra sugeneruos PU1 kai bus naujų duomenų. Kai įvyksta PU1 pertraukimas, iškviečiama pertraukimų paprogramė, kuri nuskaito duomenis iš 0x60 įvedimo/išvedimo prievado adreso. Šie duomenys – tai klaviatūros įvesties kodas. Šiame pavyzdyje patikrinama, ar klavišas buvo paspaustas ar atleistas. Jei jis buvo paspaustas, gautas kodas paverčiamas į ASCII kodą ir simbolis išvedamas į ekraną. Taip pat reikia parašyti

„keyboad_install“ funkciją, kuri iškviečia „irq_install_handler“, kad būtų galima įdiegti paprogramę „keyboard_handler“ vietoj standartinės nustatytos PU1. **[main.c]** reikia iškviesti „keyboard_install“ funkciją.

3.10.2.2 Rašymas į klaviatūros valdymo registrą

Pateikiama informacija, kaip realizuoti klaviatūros šviesų įjungimą ir išjungimą. Norint tai padaryti, reikia nusiųsti komandą į klaviatūros valdiklį. Tam yra apibrėžta specifinė procedūra. Visų pirma, reikia palaukti kol valdiklis praneš, kad jis yra laisvas. Tai padaroma nuskaitant valdymo registrą (kai skaitoma iš jo, jis vadinamas būsenos registru) cikle, kuris nutraukiamas kai klaviatūra laisva:

```
    if ((inportb(0x64) & 2) == 0) break;
```

Po ciklo galima rašyti komandos baitą į duomenų registrą. Tik tam tikrai atvejais yra rašoma tiesiai į valdymo registrą. Klaviatūros šviesų nustatymui iš pradžių siunčiamas komandos baitas 0xED, vėliau siunčiamas baitas nusakantis kokios šviesos turi būti įjungtos arba išjungtos. Baito struktūra: pirmas bitas yra „scrolllock“, antras – „numlock“, trečias – „capslock“.

Tai realizavus gaunama esminis klaviatūros palaikymas, kurį pagal poreikius galima praplėsti. Be to klaviatūros valdiklis yra taip pat naudojamas ir PS/2 pelės valdymui: pagalbinis valdiklio kanalas sujungtas su PS/2 pele.

4 Rezultatai ir išvados

Susisteminus operacinių sistemų praktinių užduočių organizavimo metodikas, buvo išskirtos trys grupės: sistemos kūrimas pasitelkiant aparatūrinės įrangos simuliatorius, operacinės sistemos analizė ir modifikavimas bei sistemos kūrimas egzistuojančiai architektūrai.

Darbe apibrėžti metodikų vertinimo kriterijai: trukmė, kaina ir operacinės sistemos funkcionalumo aspektų kiekis. Remiantis jais įvertinta kiekviena metodika.

Atsižvelgiant į esamų metodikų trūkumus ir privalumus, buvo pasiūlytos dvi naujos alternatyvos: sistemos kūrimas surinkimo metodika ir kompiuterio architektūros abstrahavimo metodika. Abi jos buvo įvertintos ir palygintos su kitomis metodikomis, tokiu būdu buvo išrinkta geriausia alternatyva – operacinės sistemos kūrimas kompiuterio architektūros abstrahavimo metodika.

Darbe pateikta pasirinktąją metodiką remianti medžiaga:

1. apibrėžti operacinių sistemų kūrimo įrankių parinkimo kriterijai;
2. parinkti kriterijus atitinkantys įrankiai;
3. paruošti įrankių konfigūravimo aprašai;
4. paruošta medžiaga kompiuterio architektūros veikimo įsisavinimui – žingsninis vadovas, kaip suprogramuoti su aparatūra tiesiogiai susijusį sistemos funkcionalumą.

Studentas, pasinaudojęs čia pateikiama medžiaga, turėtų sugebėti greitai perprasti programavimo realios architektūros aparatūrinei įrangai pagrindinius principus ir turėti tvirtą pagrindą tolesniam operacinės sistemos kūrimo darbui.

Taigi, pasiekti šie rezultatai:

1. atliktas universitetinių informatikos krypties operacinių sistemų praktinių užduočių organizavimo metodikų sisteminimas;
2. pasiūlytos dvi alternatyvos egzistuojančioms praktinių užsiėmimų organizavimo metodikoms;
3. apibrėžti metodikų vertinimo kriterijai;
4. įvertintos metodikas;
5. pateikta pasirinktą alternatyvą remianti medžiagą.

Sėkmingai įvykdžius šiuos uždavinius buvo pasiektas darbo tikslas – ištirtos operacinių sistemų dalyko praktinių užsiėmimų organizavimo metodikos, pasiūlytos dvi naujos alternatyvos ir geriausią iš jų remianti medžiaga.

5 Sąvokų žodynas ir santrumpų sąrašas

Čia pateiktas darbe naudojamų santrumpų sąrašas su paaiškinimas. Taip pat pateikti raktiniai žodžiai bei jų angliški atitikmenys.

- GNU – GNU‘s Not Unix;
- GCC – GNU compiler collection;
- NASM – the netwide assembler;
- GRUB – grand unified bootloader;
- ASCII – American standard code for information interchange;
- bazinė įvesties/išvesties sistema – angl. basic input/output system (BIOS);
- globalių deskriptorių lentelė (GDL) – angl. global descriptor table (GDT);
- pertraukimų deskriptorių lentelė (PDL) – angl. interrupt descriptor table (IDT);
- pertraukimų apdorojimo paprogramė (PAP) – angl. interrupt services routines (ISR);
- pertraukimų užklausa (PU) – angl. interrupt request (IRQ);
- programuojamas pertraukimų valdiklis (PPV) – angl. programmable interrupt controller (PIC);
- video grafikos komplektas (VGK) – angl. video graphics array (VGA);
- pradinis įkrovos takelis – angl. master boot record (MBR);
- OS – operacinė sistema;
- kodo segmentas – angl. code segment (CS);
- duomenų segmentas – angl. data segment (DS);
- doroklė – angl. handler;
- užduočių būsenų segmentas – angl. task status segment;
- adresų transliatorius – angl. linker;
- paleidimo posistemė – angl. boot loader;
- disko atvaizdas – angl. image;
- kodo konstravimas – angl. build;
- konsolė – angl. terminal;
- klaviatūros įvesties kodas – angl. scancode;
- priėjimas – angl. access;
- granuliavimas – angl. granularity;
- tolimas šuolis – angl. far jump;
- žymė – angl. label;

- nuoseklioji jungtis – angl. serial port;
- lustas – angl. chip;
- valdantysis – angl. master;
- pagalbinis – angl. slave;
- įskiepis – angl. plug-in;
- diskelių įrenginys – angl. floppy.

6 Žymėjimas

- failų bei katalogų pavadinimai:
  - **[failas.pvz];**
  - **[/katalogas/].**
- komandos vykdomos terminale:

sudo aptitude install pvz

- pastabos:
  - *rašomos pasvirusiu šriftu.*
- skaičiai žymimi:
  - dešimtainiai skaičiai rašomi įprastu formatu (123.456,00);
  - šešioliktainėje sistemoje skaičiai prasideda „0x“ (0xA2B);
  - dvejetainėje sistemoje skaičiai baigiasi „b“ (01011101b).

7 Šaltinių sąrašas

[AS02] B. Atkin, E. G. Sirer. PortOS: An Educational Operating System for the Post-PC Environment, ACM Thirty-third Annual SIGCSE Technical Symposium, Kentucky, 2002, pp. 116-120.

[BA01] F. J. Ballesteros, S. Arevalo. Using inferno in advanced OS course. [žiūrėta 2008.04.02]. Prieiga per internetą: <http://www.vitanuova.com/news/inferno_aos.pdf>

[Brh09] Brho. Installing GRUB on a Hard Disk Image File. [žiūrėta 2009-03-16]. Prieiga per internetą: <http://www.omninerd.com/articles/Installing_GRUB_on_a_Hard_Disk_Image_File>

[CF88] D. Comer, T. Fossum. Operating System Design, The Xinu Approach , Prentice Hall, New Jersey, Inc 1988, p. 496.

[DG02] R. Davoli, M. Goldweber. New Directions in Operating Systems Courses Using Hardware Simulators. [žiūrėta 2008-04-02]. Prieiga per internetą: <http://www.scs.org/getDoc.cfm?id=1957>

[DGJ09] V. Dagienė, G. Grigas, T. Jevsikova. Anglų–lietuvių kalbų kompiuterijos žodynėlis. [žiūrėta 2009-05-15]. <http://www.likit.lt/en-lt/angl.html>

[Dij01] E. W. Dijkstra. My Recollections of Operating Systems Design. [žiūrėta 2008-04-02]. Prieiga per internetą: <http://www.cs.utexas.edu/~EWD/transcriptions/EWD13xx/EWD1303.html>

[DSX05] W. Du, M. Shang, H. XuA. Novel Approach for Computer Security Education Using Minix Instructional Operating System. Syracuse University, Syracuse, 2005, p. 14.

[Fre09a] Freebyte. Free C++ (and C) Programming Tools. [žiūrėta 2009-05-05]. Prieiga per internetą: <http://www.freebyte.com/programming/cpp/#cppcompilers>

[Fre09b] Freebyte. Free Assembler programming [žiūrėta 2009-05-05]. Prieiga per internetą: <http://www.freebyte.com/programming/assembler/#freeassemblyides>

[Frei08] D. G. Feitelson. Notes on Operating Systems, The Hebrew University, Jerusalem, 2008, p. 306.

[Gon94] H. Gongzhu. A Simulated Hardware for an Operating System Course Project, Computer Science Education, Volume 5, Issue 1, 1994 , pp. 45-62.

[Int08] Intel. Intel® 64 and IA-32 Architectures Application Note TLBs, Paging-Structure Caches, and Their Invalidation. [žiūrėta 2009-04-01]. Prieiga per internetą: <http://www.intel.com/Assets/PDF/manual/317080.pdf>

[Int09a] Intel. Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 1: Basic Architecture. [žiūrėta 2009-04-01]. Prieiga per internetą: <http://www.intel.com/Assets/PDF/manual/253665.pdf>

[Int09b] Intel. Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M. [žiūrėta 2009-04-01]. Prieiga per internetą: <http://www.intel.com/Assets/PDF/manual/253666.pdf>

[Int09c] Intel. Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z. [žiūrėta 2009-04-01]. Prieiga per internetą: <http://www.intel.com/Assets/PDF/manual/253667.pdf>

[Int09d] Intel. Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide. [žiūrėta 2009-04-01]. Prieiga per internetą: <http://www.intel.com/Assets/PDF/manual/253668.pdf>

[Kov03] V. P. Kovacs. How to write a TeleType (tty) device – Part I. [žiūrėta 2009-04-03] . Prieiga per internetą: <http://osdev.berlios.de/tty.html>

[Lat02] L. Latour. Operating Systems Semester Projects. [žiūrėta 2008-04-02]. Prieiga per internetą: <http://www.umcs.maine.edu/~larry/latour/courses/431/f02/projects02.html>

[MD99] M. Morsiani, R. Davoli. Learning operating system structure and implementation through the MPS computer system simulator. ACM SIGCSE Bulletin, Volume 31 , Issue 1, March 1999, pp. 63-67.

[PR08] P. Panavas, A. Rožėnas. Modelinė operacinė sistema, Vilnius, 2008, psl.38.

[Rob02] Tim Robinson. Writing a Kernel in C. [žiūrėta 2009-03-14] . Prieiga per internetą: <http://osdev.berlios.de/ckernel.html >

[Sha07] O. P. Sharma. Enhancing OS Course Using A Comprehensive Project. Journal of Computing Sciences in Colleges, Volume 22, Issue 3, January 2007, pp. 206-213.

[She99] C. K. Shene. Multithreaded Programming Can Strengthen an Operating systems course. Computer Science Education Journal, Vol. 12, December 2002, pp. 275-299.

[Sta08] R. Stallman. Why Gnu Linux. [žiūrėta 2009-04-02]. Prieiga per internetą: <http://www.gnu.org/gnu/why-gnu-linux.html>

[Šia03] G. Šiaulys. Mokomoji operacinė sistema. Vilnius, 2003, psl.41.

[TW06] A. S. Tanenbaum, A. Woodhull. Operating Systems: Design and Implementation. Prentice Hall, New Jersey, January 04, 2006, p. 1080.

8 Priedai

8.1 Kompaktinio disko turinys

Šiame skyriuje aprašyta kompaktinio disko struktūra. Diskas pateikiamas kartu su darbu.

- /doc/bakalauro_baigiamasis_darbas.doc – elektroninė šio darbo versija;
- /doc/bakalauro_baigiamasis_darbas.pdf – elektroninė šio darbo versija PDF formatu;
- /extra/boot/ - GRUB paleidimo posistemės failai;
- src/ - projekto išeities tekstai bei kiti reikalingi failai:
    - MyOS_begin – esminiai failai, su kuriais galima pradėti kurti savo operacinę sistemą pasinaudojant darbo medžiaga: link.ld, makefile, myos.img, main.c, start.asm, system.h;
    - MyOS_full – visa projekto realizacija: tie patys failai kaip ir MyOS_begin ir dar: gdt.c, isrs.c, idt.c, irq.c, kbd.c, scrn.c, timer.c.

8.2 Ubuntu diegimo galimybės

Žmonės naudojantys kitas operacines sistemas, pavyzdžiui, Windows, greičiausiai nenorės jos pašalinti ir vietoje jos įdiegti GNU/Linux. Tačiau yra kitų būtų, kaip pasinaudoti Ubuntu sistema ir šiame darbe pateikiama informacija. Tam net nebūtina fiziškai įdiegti sistemą į kietąjį diską. Keli Ubuntu naudojimo variantai:

- Ubuntu paleidimas tiesiai iš kompaktinio disko (Ubuntu Live CD);
- diegimas į usb atmintinę/išorinį kietąjį diską;
- diegimas į kompiuterio emuliatorių (pvz., VirtualBox, Bochs, VirtualPC).

Lengviausias variantas paleisti iš kompaktinio disko, tačiau kompiuterio išjungimo metu visos naujai įdiegtos programos bus pašalintos. Kitą vertus, diegimas į usb atmintinę šią problemą išsprendžia, bet tai gali būti pakankamai sudėtinga. Jei yra galimybė, geriausia diegti į išorinį kietąjį diską. Dar viena galimybė – diegti sistemą į emuliatorių.

8.2.1 Diegimas

Pasirinkta buvo naudoti VirtualBox Intel x86 architektūros emuliatorių, kuris šiam tikslui labai gerai tinka ir yra nemokamas įrankis. Parsisiųsti VirtualBox galima tiesiai iš namų puslapio (< www.virtualbox.org >). Paleidus programą, spausti „New“ ir tiesiog sekti tolimesnes instrukcijas. Tikrai paprasta įsidiegti norimą operacinę sistemą, šiuo atveju Ubuntu. Jei kyla kažkokių neaiškumų, reikalingą dokumentaciją galima rasti namų puslapyje.

Taip pat patartina įdiegti „Guest Additions“ paketą į VirtualBox emuliatoriuje įdiegtą operacinę sistemą. Šis paketas leidžia panaudoti daug darbo patogumo patobulinimui skirtų funkcijų, tokių kaip pavyzdžiui, lango dydžio keitimas pele, realios sistemos katalogų pasiekimas ir kt. Paleidus operacinę sistemą programoje pasirinkti „Devices“ -> „Install Guest Additions...“ ir sekti tolimesnes instrukcijas.

Taigi, pasitelkus VirtualBox emuliatorių, galima įdiegti Ubuntu (ar kitą GNU/Linux), pasinaudoti darbe pateikiama medžiaga, išbandyti visus nurodytus įrankius ir išsaugoti kompiuterio operacinę sistemą visiškai nepaliestą.

8.3 Išeities tekstų ištraukos

```
     /* Nuoroda į teksto atmintį,  teskto ir fono spalvos,  x ir y kursorių
koordinatės. Globalūs failo „scrn.c“ kintamieji */
     unsigned short *textmemptr;
     int attrib = 0x0F;
     int csr_x = 0, csr_y = 0;

     /* Paslenka ekraną */
     void scroll(void)
     {
         unsigned blank, temp;

         /* Tuščia vieta yra tarpas, kuris gali turėti fono spalvą */
         blank = 0x20 | (attrib << 8);

         /* 25 eilutė paskutė, reiškia reikia paslinkti į viršų */
         if(csr_y >= 25)
         {
             /* Patraukia dabartinį tekstą viena eilute atgal */
             temp = csr_y - 25 + 1;
             memcpy ((unsigned char *)textmemptr, (unsigned char *)(textmemptr +
temp * 80), (25 - temp) * 80 * 2);

             /* Paskutinę eilutę užpildom tarpais */
             memsetw (textmemptr + (25 - temp) * 80, blank, 80);
             csr_y = 25 - 1;
         }
     }
```

22 C kodo ištrauka – [scrn.c]

```
     /* Įdiega PDL */
```

```
void idt_install()
{
    /* Sukuria specialią nuorodą, reikalingą PDL užkrovimui.
     * Identiška GDL nuorodai. */
    idtp.limit = (sizeof (struct idt_entry) * 256) - 1;
    idtp.base = &idt;

    /* Visus PDL įrašus „nunulina“ */
    memset(&idt, 0, sizeof(struct idt_entry) * 256);

    /* Čia galima įtraukti naujas pertraukimų apdorojimo paprogrames,
     * naudojantis funkcija idt_set_gate */

    /* Nustato procesoriaus registrą į PDL */
    idt_load();
}
```

23 C kodo ištrauka – [idt.c]

```
; Iškviečia funkciją esančią faile „irc.c“
irq_common_stub:
    pusha
    push ds
    push es
    push fs
    push gs
    mov ax, 0x10
    mov ds, ax
    mov es, ax
    mov fs, ax
    mov gs, ax
    mov eax, esp
    push eax
    mov eax, irq_handler
    call eax
    pop eax
    pop gs
    pop fs
    pop es
    pop ds
    popa
    add esp, 8
```

```
        iret
```

6 ASM kodo ištrauka – [start.asm]