Ինֆորմատիկա, հաշվողական տեխնիկայի և
Կառավարման համակարգերի ամբիոն
Հաշվողական տեխնիկայի և
կառավարման Ֆակուլտետ

ԴԻՊԼՈՄԱՅԻՆ ԱՇԽԱՏԱՆՔ

ՀԿ-92 խմբի ուսանող

Սարգսյան Վրույր Վախթանգի

(Ազգանուն, անուն, հայրանուն)

Թեմա՝

Աշխատանք IRC RFC 1459-ի հետ: IRC-ի նախագծում

ԵՐԵՎԱՆ 2024թ.

Բովանդակություն

Խնդրի դրվածք

Դիպլոմային աշխատանքի նպատակն է նախագծել և իրականացնել IRC RFC 1459 ծրագրային մոդուլը: Մշակցված ծրագրային համակարգը հնարավորություն կտա օգտվողներին հաղորդակցվել իրար հետ, ստեղծել խմբեր և ուղարկել միմյանց նամակներ:

Աշխատանքի ընթացքում անհրաժեշտ է.

- Ունենալ աշխատող և անխափան սերվեր,
- Ստեղծել կլիենտների միջև կապ,
- Սոքեթներ և պրոտոկոլներ,
- Հրամաններ ստեղծել խումբը կառավարողի համար,
- Իմպլեմենտացնել կլիենտ-սերվեր կապը RFC 1459 ստանդարտին համաձայն:

Ներածություն

IRC (Internet Relay Chat) համակարգչային ցանցերի միջոցով իրական ժամանակում տեքստային հաղորդագրությունների փոխանցման արձանագրություն է: Դրա ստեղծման պատճառ է դարձել ամբողջ աշխարհում

բաշխված օգտատերերի միջև արդյունավետ և ակնթարթային հաղորդակցության անհրաժեշտությունը: 1988 թվականին ֆինն մշակող Յարկկո Օիկարինենը սկսեց աշխատել մի նախագծի վրա, որը ներառում էր հաղորդագրությունների փոխանակում օգտատերերի միջև կենտրոնացված սերվերի միջոցով: Այս գաղափարը ծագել է ծրագրավորողների և գիտնականների համայնքում հաղորդակցության անհրաժեշտությունից՝ հեշտացնելու փորձի և գիտելիքների փոխանակումը: IRC արձանագրությունը ապահովում է ալիքներ ստեղծելու հնարավորություն, որտեղ մասնակիցները կարող էին քննարկել որոշակի թեմաներ: Հիմնական կետը արձանագրության ստանդարտացումն էր, որը տեղի ունեցավ 1993 թվականին RFC 1459-ի թողարկմամբ: Այս փաստաթուղթը սահմանեց IRC-ի միջոցով հաղորդագրություններ ուղարկելու հիմնական կանոններն ու ձևաչափերը:

Ինտերնետի զարգացման և համաշխարհային համակարգչային ցանցերի առաջացման հետ մեկտեղ IRC-ն դարձել է հաղորդակցության լայն տարածում ունեցող միջոց: Այն գրավել է տարբեր համայնքների ուշադրությունը, ինչպիսիք են խաղացողները, ծրագրեր մշակողները, արվեստագետները և շատ ավելին: Յուրաքանչյուրը կարող էր ստեղծել իր սեփական ալիքը և հրավիրել մասնակիցներին քննարկելու իրեն հետաքրքրող թեմաները:

IRC-ն իրական ժամանակում հաղորդակցվելու և մտքեր փոխանակելու ազատություն է տվել՝ այն հանրաճանաչ դարձնելով ընդհանուր շահեր ունեցող համայնքներում: Այս արձանագրությունը չէր ենթադրում կենտրոնացված հսկողություն, և յուրաքանչյուրը կարող էր ստեղծել իր սեփական սերվերը և ինքնուրույն կառավարել այն: Այս բաց և ապակենտրոնացված սարքը IRC-ն դարձրել է հաղորդակցման եզակի գործիք:

Չնայած կապի ավելի ժամանակակից միջոցների առաջացմանը, ինչպիսիք են ակնթարթային հաղորդագրությունները և սոցիալական ցանցերը, IRC-ն շարունակում է տարածված մնալ որոշ շրջանակներում՝ պահպանելով իր տեղը

ինտերնետ կապի պատմության մեջ և այլընտրանք տրամադրելով ավելի կառուցվածքային և կենտրոնացված հարթակներին:

Գլուխ 1

1.1 Նկարագրություն

IRC (Internet Relay Chat) արձանագրությունը նախագծվել է տեքստի վրա հիմնված կոնֆերանսների օգտագործման համար: Այս փաստաթուղթը նկարագրում է ընթացիկ IRC արձանագրությունը:

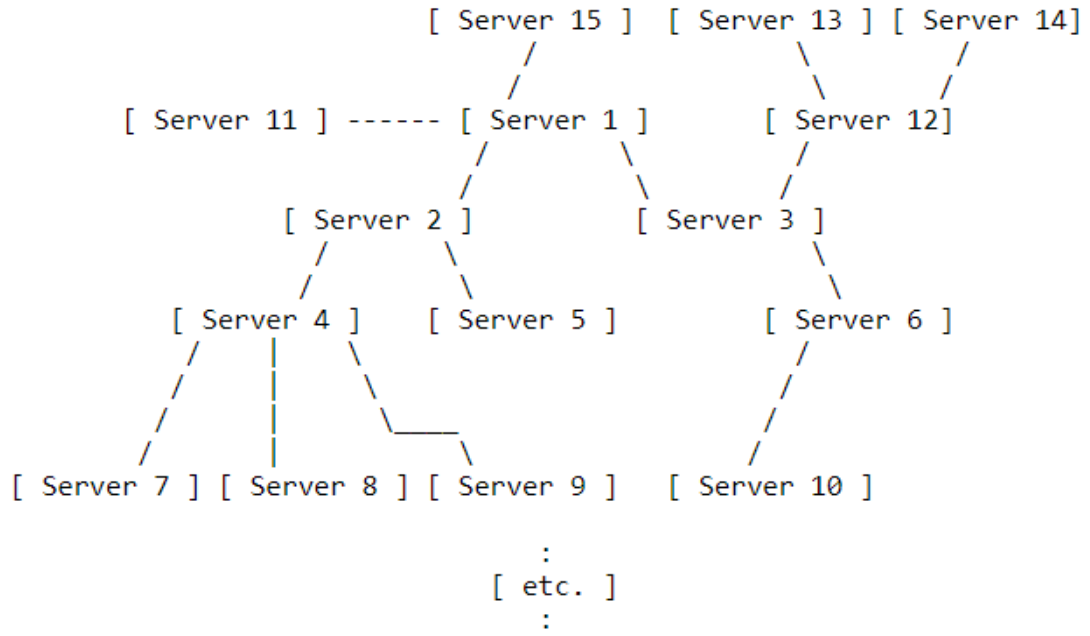
IRC արձանագրությունը մշակվել է TCP/IP օգտագործող համակարգերի համար: IRC-ն ինքնին հեռահաղորդակցության համակարգ է, որը (օգտագործելով հաճախորդ-սերվեր մոդելը) հարմար է բազմաթիվ մեքենաների վրա աշխատելու համար բաշխված ձևով. Տիպիկ կարգավորումը ներառում է մեկ գործընթաց (սերվերը), որը կենտրոնական կետ է կազմում հաճախորդներին(կամ այլ սերվերների) միանալու համար՝ կատարելով անհրաժեշտ հաղորդագրությունների փոխանակում և այլ գործառույթներ:

1.1 Սերվեր

Սերվերը կազմում է IRC-ի ողնաշարը՝ տրամադրելով մի կետ, որին

հաճախորդները կարող են միանալ միմյանց հետ խոսելու համար, և մի կետ սերվերներ միանալու համար՝ ձևավորելով IRC ցանց: Միակ ցանցը IRC սերվերների համար թույլատրված կոնֆիգուրացիան ընդգրկող ծառի կոնֆիգուրացիան է [Նկար 1] որտեղ յուրաքանչյուր սերվեր գործում է որպես կենտրոնական հանգույց մնացած մասերի համար:

Նկար 1.



1.2 Կլեինտ (հաճախորդ)

Հաճախորդը այն ամենն է, որը միանում է սերվերին եթե դա սերվեր չէ: Յուրաքանչյուր հաճախորդ տարբերվում է մյուս հաճախորդներից

յուրահատկությամբ մականուն (nickname), որն ունի առավելագույն երկարություն ինը (9) նիշ:
Բոլոր կանոնները կարող եք գտնել համացանցում և ուսումնասիրել թե ինչ է կարելի և թե ինչ չի կարելի օգտագործել որպես մականուն: Բացի մականունից, բոլոր սերվերները պետք է ունենան բոլոր հաճախորդների մասին հետևյալ տեղեկությունները: Հյուրընկալողի (host) անունը որը հիմնականում առաջին կլիենտն է և սերվերի անունն, որին միացված է հաճախորդը:

1.3 Օպերատորներ

IRC ցանցում պատվերների ողջամիտ ծավալի պահպանումն ապահովելու համար հաճախորդների հատուկ դասին (օպերատորներին) թույլատրվում է կատարել ընդհանուր սպասարկման գործառույթները ցանցում: Չնայած լիազորությունները տրամադրված օպերատորին կարող է դիտվել որպես "վտանգավոր", նրանք այնուամենայնիվ անհրաժեշտ են: Օպերատորները պետք է կարողանան կատարել ցանցի հիմնական խնդիրները ինչպես օրինակ անջատում և վերամիացում սերվերների, ինչպես նաև պետք է կանխել երկարաժամկետ օգտագործումը:
Ընդունելով այս անհրաժեշտությունը ՝ Այստեղ քննարկվող արձանագրությունը նախատեսում է, որ օպերատորները կարողանան կատարել միայն այդպիսի գործառույթներ: Օպերատորների ավելի հակասական առավելությունը հնարավորությունն է
"ստիպել (force)" օգտագործողին հեռացնել (kick) միացված ցանցից, այսինքն ՝ օպերատորները կարող են փակել կապը ցանկացած հաճախորդի և սերվերի միջև: Դրա հիմնավորումը նուրբ է, քանի որ դրա չարաշահումը միաժամանակ կործանարար և նյարդայնացնող է:

1.3 Ալիքներ (Channels)

Ալիքը մեկ կամ մի քանի հաճախորդների անվանված խումբ է, որտեղ բոլորը կարող են ստանալ հաղորդագրություններ՝ ուղղված այդ ալիքին: Ալիքը ստեղծված է անուղղակիորեն, երբ առաջին հաճախորդը միանում է ալիքը բացվում է, և ալիքը դադարում է գոյություն ունենալ, երբ վերջին հաճախորդը թողնում է այն: Մինչ ալիքը գոյություն ունի, ցանկացած հաճախորդ կարող է հղում կատարել ալիքին՝ օգտագործելով ալիքի անունը:

Ալիքների անունները տողեր են (սկսվող «&» կամ «#» գրանշաններով):
երկարությունը մինչև 200 նիշ: Բացի այն պահանջից, որ
առաջին նիշը կամ '&' կամ '#'; միակ սահմանափակումը ա
ալիքի անվանումն այն է, որ այն չի կարող պարունակել բացատներ (' '),
վերահսկիչ G (^G կամ ASCII 7), կամ ստորակետ (','), որն օգտագործվում է որպես ցանկի տարր բաժանարար արձանագրությամբ):
Այս արձանագրությամբ թույլատրված են երկու տեսակի ալիքներ:

Բաշխված ալիք, որը հայտնի է ցանցին միացած բոլոր սերվերներին: Այս ալիքները նշվում են առաջինով նիշը լինելով միակ հաճախորդը սերվերի վրա, որտեղ այն կա, հաճախորդները կարող են միանալ. Սրանք առանձնանում են առաջատար «&» նշանով: Այս տեսակների համար կան ալիքի տարբեր ռեժիմներ:
Նոր ալիք ստեղծելու կամ գոյություն ունեցող ալիքի, օգտատեր դառնալու համար ալիքին միանալը պարտադիր է: Եթե ալիքը նախկինում գոյություն չունի

միանալու համար ալիքը ստեղծվում է, և ստեղծող օգտատերը դառնում է ալիքի օպերատոր: Եթե ալիքն արդեն գոյություն ունի, անկախ նրանից՝ ձերն է, թե ոչ:

Միանալու հայցը, կախված է ընթացիկ ռեժիմներից ալիքի:

Օրինակ, եթե ալիքը նախատեսված է միայն հրավիրելու համար, (+i), ապա կարող եք միանալ միայն հրավիրված լինելու դեպքում: Որպես արձանագրության մաս՝ օգտատեր կարող է լինել միանգամից մի քանի ալիքում, բայց տասը (10) սահմանաչափով:

1.4 Ալիքի օպերատոր (The channel operator)

Ալիքի օպերատորը համարվում է այդ ալիքի «սեփականատեր»:

Այս կարգավիճակով օպերատորներն օժտված են որոշակի լիազորություններով, որոնք նրանց հնարավորություն են տալիս իրենց ալիքում վերահսկողություն և որոշակի ողջախոհություն պահպանել:

Որպես ալիքի սեփականատեր, ալիքի օպերատորը պարտավոր չէ ունենալ պատճառները իրենց գործողությունների համար, թեև եթե նրանց գործողությունները ընդհանուր են հակասոցիալական կամ այլ կերպ վիրավորական, կարող է հեռանալ և գնալ այլ տեղ և ձևավորել իրենց սեփական ալիքը:

Հրամանները, որոնք կարող են օգտագործվել միայն ալիքի օպերատորների կողմից, հետևյալն են.

KICK - Հեռացնել հաճախորդին ալիքից

MODE - Փոխեք ալիքի ռեժիմը

INVITE - Միայն հրավերով ալիք (+i)

TOPIC - Փոխել ալիքի թեման +t

Ալիքի օպերատորը նույնացվում է «@» նշանով, որը գտնվում է նրանց կողքին մականունը, երբ այն կապված է ալիքի հետ (այսինքն՝ պատասխանում է NAMES, WHO և WHOIS հրամաններին):

1.5 Նամակներ

Սերվերները և հաճախորդները միմյանց հաղորդագրություններ են ուղարկում, որոնք (կարող են կամ ոչ) առաջացնել պատասխան. Եթե հաղորդագրությունը պարունակում է վավեր հրաման, հաճախորդը պետք է ակնկալի պատասխան, ինչպես նշված է, բայց խորհուրդ չի տրվում հավերժ սպասել պատասխանին: Հաճախորդ - սերվեր և սերվերից - սերվերի միջև հաղորդակցությունը հիմնականում կրում է ասինխրոն բնույթ.

Յուրաքանչյուր IRC հաղորդագրություն կարող է բաղկացած լինել մինչև երեք հիմնական մասից՝ նախածանցից (ըստ ցանկության), հրամանը և հրամանի պարամետրերը (որը կարող է լինել մինչև 15 նշան): Նախածանցը, հրամանը և բոլոր պարամետրերը առանձնացված մեկ (կամ մի քանի) ASCII տարածության նիշ(ներ)ով (ASCII Hex 0x20) :

Նախածանցի առկայությունը նշվում է մեկ առաջատար ASCII-ով
երկու կետի նշան (':', 0x3b):

1.6 IRC ճարտարապետություն

Ինտերնետ ռելե զրույցի (IRC) ճարտարապետությունը հիմնարար նշանակություն ունի՝ հասկանալու համար, թե ինչպես է գործում արձանագրությունը և հեշտացնում է օգտատերերի միջև հաղորդակցությունը: IRC-ն աշխատում է ապակենտրոնացված հաճախորդ-սերվեր մոդելի վրա, որը շատ կարևոր է դրա մասշտաբայնության և ճկունության համար:

Սերվերի ենթակառուցվածք
IRC ցանցերը բաղկացած են փոխկապակցված սերվերներից, որոնցից յուրաքանչյուրը աշխատում է IRC սերվերի ծրագրակազմով: Այս սերվերները կազմում են IRC ցանցի ողնաշարը՝ հեշտացնելով հաղորդագրությունների փոխանակումը հաճախորդների միջև և պահպանելով ամբողջ ցանցի համաժամացումը:

Հաճախորդների միացումներ
Հաճախորդները միանում են IRC սերվերներին՝ օգտագործելով մասնագիտացված IRC հաճախորդի ծրագրակազմ: Միանալուց հետո հաճախորդները կարող են միանալ ալիքներին, մասնակցել մասնավոր զրույցներին և շփվել ցանցի այլ օգտատերերի հետ:
Ալիքները վիրտուալ տարածքներ են IRC ցանցում, որտեղ օգտվողները կարող են իրական ժամանակում զրուցել միմյանց հետ: Յուրաքանչյուր ալիք ունի յուրահատուկ անուն և ծառայում է որպես կոնկրետ թեմաներ կամ հետաքրքրություններ քննարկելու հարթակ:

Ապակենտրոնացված հաղորդակցություն

IRC-ի հիմնական առանձնահատկություններից մեկը նրա ապակենտրոնացված հաղորդակցման մոդելն է: Հաղորդագրությունները փոխանցվում են սերվերների միջև հավասարազոր ձևով, ինչը թույլ է տալիս հաղորդագրությունների արդյունավետ բաշխումը ցանցում: Այս ապակենտրոնացված ճարտարապետությունը նպաստում է IRC-ի կայունությանը և սխալների հանդուրժողականությանը:

Ընդարձակություն և հուսալիություն

IRC-ի ապակենտրոնացված ճարտարապետությունը թույլ է տալիս այն նրբագեղորեն մասշտաբավորել մեծ թվով օգտվողների և ալիքների տեղավորելու համար: Սերվերները կարելի է դինամիկ կերպով ավելացնել կամ հեռացնել ցանցից՝ ապահովելով շարունակական հասանելիություն և հուսալիություն նույնիսկ բարձր բեռների դեպքում:

1.7 IRC հաղորդակցություն

Հաղորդակցությունը գտնվում է Internet Relay Chat-ի (IRC) հիմքում, որը հնարավորություն է տալիս օգտվողներին փոխանակել հաղորդագրությունները, միանալ ալիքներին և մասնակցել զրույցներին իրական ժամանակում: Հասկանալը, թե ինչպես է հաղորդակցությունը տեղի ունենում IRC-ում, կարևոր է դրա դինամիկան և ֆունկցիոնալությունը հասկանալու համար:

Հաղորդագրության անցում

Իր հիմքում IRC հաղորդակցությունը պտտվում է հաճախորդների և սերվերների միջև հաղորդագրությունների փոխանակման շուրջ: Երբ օգտվողը հաղորդագրություն է ուղարկում, այն հաճախորդի կողմից փոխանցվում է IRC սերվերին, որն այնուհետև այն բաշխում է համապատասխան հասցեատերերին:

Սերվեր-հաճախորդ փոխազդեցություն

Սերվերները կենտրոնական դեր են խաղում հաճախորդների միջև հաղորդակցությունը հեշտացնելու գործում: Նրանք մշակում են մուտքային հաղորդագրությունները, ուղղորդում դրանք դեպի իրենց նախատեսված նպատակակետերը և պահպանում են պետական տեղեկատվությունը, ինչպիսիք են օգտատերերի կարգավիճակները և ալիքի անդամակցությունները:

Ալիքի վրա հիմնված հաղորդակցություն

Ալիքները ծառայում են որպես կապի հիմնական միջոց IRC-ում: Օգտատերերը կարող են միանալ ալիքներին՝ մասնակցելու խմբային քննարկումներին կամ ստեղծել իրենց սեփական ալիքները կոնկրետ թեմաների կամ հետաքրքրությունների համար: Ալիք ուղարկված հաղորդագրությունները հեռարձակվում են այդ ալիքի բոլոր անդամներին:

Իրական ժամանակի փոխազդեցություններ

IRC-ի որոշիչ առանձնահատկություններից մեկը իրական ժամանակի փոխազդեցություններին աջակցությունն է: Հաղորդագրությունները հասցվում են հասցեատերերին գրեթե ակնթարթորեն, ինչը թույլ է տալիս հոսուն և դինամիկ խոսակցություններ օգտատերերի միջև:

Ցանցի համաժամացման

IRC սերվերները միմյանց միջև փոխանակում են հաղորդագրություններ՝ համաժամեցնելու օգտատերերի կարգավիճակները, ալիքների

անդամակցությունները և ցանցի ամբողջ տեղեկատվությունը: Սա ապահովում է հետևողականություն և հետևողականություն ամբողջ IRC ցանցում:

1.8 Անվտանգության նկատառումներ

Անվտանգությունը գերակա խնդիր է Internet Relay Chat-ում (IRC)՝ դրա բաց և ապակենտրոնացված լինելու պատճառով: IRC ցանցերը խոցելի են անվտանգության տարբեր սպառնալիքների նկատմամբ, ներառյալ չարտոնված մուտքը, տվյալների գաղտնալսումը և վնասակար գործողությունները: IRC հաղորդակցությունների ամբողջականությունը, գաղտնիությունը և հասանելիությունն ապահովելու համար օգտատերերը, սերվերի ադմինիստրատորները և IRC օպերատորները պետք է ապահովեն ամուր անվտանգության միջոցներ: Ստորև բերված են IRC-ի անվտանգության որոշ հիմնական նկատառումներ.

1. Նույնականացման մեխանիզմներ
Նույնականացման մեխանիզմները վճռորոշ դեր են խաղում օգտատերերի ինքնությունը ստուգելու և ապահովելու համար, որ միայն լիազորված անձինք կարող են մուտք գործել IRC ցանցեր և ալիքներ: Նույնականացման ընդհանուր մեթոդները ներառում են.

NickServ. IRC շատ ցանցեր տրամադրում են NickServ ծառայություն, որը թույլ է տալիս օգտվողներին գրանցել իրենց մականունները և հաստատել իրենց իսկությունը՝ օգտագործելով գաղտնաբառերը: Գրանցված մականունները

ապահովում են պատասխանատվության մակարդակ և օգնում են կանխել անձը կեղծելը և չարտոնված մուտքը:
SASL (Simple Authentication and Security Layer). SASL-ը հաճախորդների համար ապահովում է IRC սերվերների միջոցով նույնականացման մեխանիզմ՝ օգտագործելով նույնականացման տարբեր մեխանիզմներ, ինչպիսիք են պարզ տեքստը, CRAM-MD5 կամ արտաքին նույնականացման արձանագրությունները, ինչպիսիք են OAuth-ը:
Նույնականացման ուժեղ գործելակերպի կիրառումն օգնում է նվազեցնել չարտոնված մուտքի վտանգը և պաշտպանում է օգտատերերի գաղտնիությունն ու ինքնությունը:

2. Գաղտնագրման արձանագրություններ
Գաղտնագրման արձանագրությունները կարևոր են IRC հաղորդակցությունների գաղտնիությունն ու ամբողջականությունը պաշտպանելու համար, հատկապես հանրային ցանցերում, որտեղ հաղորդագրությունները կարող են անցնել անվստահելի ցանցեր և սերվերներ: IRC-ում օգտագործվող գաղտնագրման ընդհանուր արձանագրությունները ներառում են.

SSL/TLS (Secure Sockets Layer/Transport Layer Security). SSL/TLS գաղտնագրումը ապահովում է հաճախորդների և սերվերների միջև կապի ալիքը՝ գաղտնագրելով տարանցիկ տվյալները և պաշտպանելով դրանք գաղտնալսումից և կեղծումից:
DCC (Direct Client-to-Client). DCC-ն օգտատերերին թույլ է տալիս ուղիղ հավասարակցական կապեր հաստատել ֆայլերի փոխանցման և մասնավոր խոսակցությունների համար: DCC միացումներում գաղտնագրման իրականացումը երաշխավորում է, որ զգայուն տվյալները մնում են գաղտնի և անվտանգ:
SSL/TLS գաղտնագրման ակտիվացումը և անվտանգ DCC կապերի օգտագործումը օգնում են կանխել IRC տրաֆիկի չթույլատրված գաղտնալսումը և պաշտպանել օգտատերերի գաղտնիությունը:

3. Մուտքի վերահսկման մեխանիզմներ

Մուտքի վերահսկման մեխանիզմները սերվերի ադմինիստրատորներին և ալիքի օպերատորներին հնարավորություն են տալիս կիրառել մուտքի քաղաքականություն և սահմանափակել օգտատերերի արտոնությունները՝ հիմնվելով նախապես սահմանված չափանիշների վրա: Մուտքի վերահսկման ընդհանուր մեխանիզմները ներառում են.

Ալիքի ռեժիմներ. IRC ալիքներն աջակցում են տարբեր ռեժիմներ, որոնք թույլ են տալիս օպերատորներին վերահսկել, թե ովքեր կարող են միանալ, խոսել և կատարել վարչական գործողություններ ալիքի ներսում: Ռեժիմները, ինչպիսիք են +i (միայն հրավիրել), +o (օպերատոր) և +b (արգելք) օգնում են պահպանել կարգը և անվտանգությունը ալիքներում:

OperServ. OperServ-ը բազմաթիվ IRC ցանցերի կողմից տրամադրվող ծառայություն է, որը թույլ է տալիս սերվերի օպերատորներին (IRCops) կատարել ադմինիստրատիվ առաջադրանքներ և կիրառել ամբողջ ցանցի քաղաքականություն, օրինակ՝ սահմանափակել մուտքը որոշակի IP տիրույթներ կամ իրականացնել ցանցի ողջ արգելք:

Մուտքի վերահսկման կայուն մեխանիզմների ներդրումն օգնում է չլիազորված օգտատերերին խափանել խոսակցությունները, սպամ ալիքները կամ չարամիտ գործողությունները:

4. Բոտի և սցենարի անվտանգություն

Բոտերի և սկրիպտների անվտանգությունը կարևոր է IRC հաճախորդները և սերվերները վնասակար սկրիպտներից, բոտերից և շահագործումներից պաշտպանելու համար, որոնք կարող են վտանգի ենթարկել համակարգի ամբողջականությունը կամ հեշտացնել չարտոնված մուտքը: Բոտի և սցենարների անվտանգության լավագույն փորձը ներառում է.

Կոդի վերանայում. Պարբերաբար վերանայեք և ստուգեք բոտի սկրիպտները և պլագինները՝ հայտնաբերելու և վերացնելու անվտանգության հնարավոր

խոցելիությունները, ինչպիսիք են հրամանի ներարկումը, բուֆերային հոսքերը կամ արտոնությունների ընդլայնումը:
Sandbox միջավայրեր. Գործարկեք բոտեր և սկրիպտներ մեկուսացված ավազատուփ միջավայրերում՝ սահմանափակ արտոնություններով՝ նվազագույնի հասցնելու անվտանգության հնարավոր միջադեպերի ազդեցությունը և կանխելու զգայուն ռեսուրսների չարտոնված մուտքը:
Պարբերաբար թարմացրեք. Պահպանեք IRC հաճախորդի ծրագրակազմը, սկրիպտները և պլագինները թարմացված անվտանգության վերջին պատչերի և թարմացումների հետ՝ նվազեցնելու համար հարձակվողների կողմից հայտնի խոցելիության վտանգը:
Հետևելով այս լավագույն փորձին, IRC օգտվողները և ադմինիստրատորները կարող են նվազագույնի հասցնել անվտանգության խախտումների ռիսկը և ապահովել IRC ցանցերի և ալիքների անվտանգ և անվտանգ շահագործումը:

1.9 Օգտագործողի և ալիքի ռեժիմներ

Internet Relay Chat-ում (IRC) օգտվողի ռեժիմները և ալիքի ռեժիմները նշանակալի դեր են խաղում ցանցի ներսում վարքագծի և փոխազդեցությունների ձևավորման գործում: Այս ռեժիմները օգտատերերին և ալիքների օպերատորներին հնարավորություն են տալիս վերահսկել իրենց IRC փորձառության տարբեր ասպեկտները, ներառյալ մուտքի թույլտվությունները, տեսանելիությունը և վերահսկման հնարավորությունները:

Օգտագործողի ռեժիմներ

IRC-ն օգտատերերին թույլ է տալիս տարբեր ռեժիմներ սահմանել իրենց վրա՝ փոխելով, թե ինչպես են նրանք փոխազդում ցանցի և այլ օգտատերերի հետ: Օգտագործողի որոշ սովորական ռեժիմներ ներառում են.

+i (Անտեսանելի). Երբ օգտվողն իրեն դնում է որպես անտեսանելի, նրանք թաքնվում են այլ օգտվողների WHOIS հարցումներից: Այս ռեժիմն ապահովում է գաղտնիություն՝ թույլ տալով օգտվողներին անանուն մնալ:
+o (Օպերատոր). Օպերատորի ռեժիմը ալիքի ներսում օգտատերերին տալիս է արտոնություններ՝ հնարավորություն տալով նրանց կատարել մոդերատորական գործողություններ, ինչպիսիք են՝ KICK կամ արգելելը այլ օգտատերերի:
+s (Սերվերի ծանուցումներ). Օգտագործողները, որոնց սերվերի ծանուցումների ռեժիմը միացված է, ստանում են կարևոր սերվերի ծանուցումներ, ինչպիսիք են սպասարկման ծանուցումները կամ կապի կարգավիճակի թարմացումները:
Օգտատիրոջ ռեժիմները հնարավորություն են տալիս անհատներին հարմարեցնել իրենց IRC փորձը և արդյունավետ կառավարել իրենց փոխազդեցությունները այլ օգտատերերի և ալիքների հետ:

1.9.1 Ալիքի ռեժիմներ

IRC ալիքներն աջակցում են տարբեր ռեժիմներ, որոնք ալիքի օպերատորները կարող են սահմանել՝ վերահսկելու մուտքը, տեսանելիությունը և վարքագիծը ալիքի ներսում: Ալիքի որոշ սովորական ռեժիմներ ներառում են.

+t (TOPIC). Թեմայի ռեժիմը թույլ չի տալիս ալիքի օպերատոր չհանդիսացող օգտատերերին փոխել ալիքի թեման: Այս ռեժիմն օգնում է պահպանել հետևողականությունն ու համապատասխանությունը ալիքի քննարկումներում:
+n (Առանց արտաքին նամակների). Արտաքին հաղորդագրությունների ռեժիմը չի սահմանափակում հաղորդագրությունները միայն ալիքի անդամ օգտվողների համար: Այս ռեժիմը թույլ չի տալիս կողմնակի անձանց խափանել խոսակցությունները և ալիքը սպամ ուղարկել:
+m (Moderated). Մոդերացված ռեժիմը թույլ է տալիս միայն ալիքի օպերատորներին և ձայնային (+v) կամ կիսաօպերատորի (+h) կարգավիճակ ունեցող օգտվողներին հաղորդագրություններ ուղարկել ալիքին: Այս ռեժիմը օգտակար է զբաղված ալիքներում խոսակցության հոսքը վերահսկելու համար:
Ալիքի ռեժիմները ալիքի օպերատորներին տալիս են ճկունություն՝ հարմարեցնելու իրենց ալիքների կարգավորումները՝ իրենց համայնքի կարիքներին համապատասխան և կանոններ կիրառելու՝ դրական և կանոնավոր միջավայր պահպանելու համար:

Օգտատիրոջ և ալիքի ռեժիմների համատեղում
Օգտվողի ռեժիմները և ալիքի ռեժիմները աշխատում են զուգահեռաբար՝ օգտատերերին և ալիքի օպերատորներին ապահովելու իրենց IRC փորձի վրա մանրակրկիտ վերահսկողություն: Սահմանելով համապատասխան ռեժիմներ՝ օգտատերերը կարող են կառավարել իրենց տեսանելիությունը, մուտքի թույլտվությունները և փոխգործակցության հնարավորությունները ցանցի և կոնկրետ ալիքների ներսում:
Մյուս կողմից, ալիքի օպերատորները կարող են կիրառել կանոններ և քաղաքականություն՝ ապահովելու իրենց ալիքների անխափան աշխատանքը և չափավորությունը:

Տեխնիկական Մանրամասնություններ

1.1 Makefile-ի բովանդակություն և բացատրություն

```
#Makefile-ը նման է բաղադրատոմսերի գրքի՝ ծրագրային նախագծերի կառավարման համար: Այն պարունակում է հրահանգներ (հայտնի են որպես կանոններ) այնպիսի առաջադրանքների համար, ինչպիսիք են կոդ կազմելը, ֆայլերը կառավարելը և նախագծի հետ կապված այլ գործողություններ: Յուրաքանչյուր կանոն ունի թիրախ (ինչ ստեղծել), կախվածություններ (ինչ է անհրաժեշտ) և հրամաններ (ինչպես դա անել): Երբ դուք գործարկում եք make-ը տերմինալում, այն կարդում է Makefile-ը և կատարում է նշված կանոնները՝ ձեր նախագիծը թարմացնելու կամ կառուցելու համար: Սա օգտակար է առաջադրանքների ավտոմատացման, հետևողականության ապահովման և բարդ նախագծերի արդյունավետ կառավարման համար:

NAME = ircserv
BOT_NAME = botserv

OBJECTS_FOLDER = ./objects/

SRCS = $(wildcard server/*.cpp)
BOT_SRCS = $(wildcard bot/*.cpp)

OBJS = $(SRCS:%.cpp=$(OBJECTS_FOLDER)%.o)
BOT_OBJS = $(BOT_SRCS:%.cpp=$(OBJECTS_FOLDER)%.o)

HEADERS = $(wildcard server/*.hpp)
BOT_HEADERS = $(wildcard bot/*.hpp)

PRE_HEADERS = $(HEADERS:%.hpp=$(OBJECTS_FOLDER)%.hpp.gch)
BOT_PRE_HEADERS = $(BOT_HEADERS:%.hpp=$(OBJECTS_FOLDER)%.hpp.gch)

CFLAGS = -Wall -Wextra -Werror -std=c++98 # -fsanitize=address -g

CC = c++

RM = rm -rf

all: objs $(PRE_HEADERS) $(NAME)

run_bot: bot_objs $(BOT_PRE_HEADERS) $(BOT_NAME)

$(OBJECTS_FOLDER)server/%.hpp.gch: server/%.hpp
    @$(CC) $(CFLAGS) $< -o $@

$(OBJECTS_FOLDER)bot/%.hpp.gch: bot/%.hpp
    @$(CC) $(CFLAGS) $< -o $@
```

```
$(OBJECTS_FOLDER)server/%.o: server/%.cpp $(PRE_HEADERS)
    @$(CC) $(CFLAGS) -c $< -o $@

$(OBJECTS_FOLDER)bot/%.o: bot/%.cpp $(BOT_PRE_HEADERS)
    @$(CC) $(CFLAGS) -c $< -o $@



$(NAME): $(OBJS)
    @$(CC) $(CFLAGS) $? -o $(NAME)

$(BOT_NAME) : $(BOT_OBJS)
    @$(CC) $(CFLAGS) $? -o $(BOT_NAME)

objs:
    @(mkdir -p objects/server)

bot_objs:
    @(mkdir -p objects/bot)

clean:
    @$(RM) $(OBJECTS_FOLDER)
    @$(RM) $(PRE_HEADERS)

fclean: clean
    @$(RM) $(NAME)

re: fclean all



test:
    @echo $(OBJS)
    @echo $(PRE_HEADERS)

.PHONY: all clean fclean re run_bot
```

NAME-ը և BOT_NAME-ը փոփոխականներ են, որոնք սահմանում են համապատասխանաբար հիմնական ծրագրի և բոտի ծրագրի անվանումները: OBJECTS_FOLDER-ը փոփոխական է, որը նշում է այն թղթապանակը, որտեղ կպահվեն օբյեկտների ֆայլերը:

SRCS-ը և BOT_SRCS-ը փոփոխականներ են, որոնք պարունակում են աղբյուրի ֆայլերի ցուցակներ սերվերի և բոտի համար:
OBJS-ը և BOT_OBJS-ը փոփոխականներ են, որոնք պարունակում են օբյեկտային ֆայլերի ցուցակներ, որոնք համապատասխանում են սերվերին և բոտի աղբյուրի ֆայլերին:
HEADERS-ը և BOT_HEADERS-ը փոփոխականներ են, որոնք պարունակում են վերնագրի ֆայլերի ցուցակներ սերվերի և բոտի համար:
PRE_HEADERS-ը և BOT_PRE_HEADERS-ը փոփոխականներ են, որոնք պարունակում են նախապես կազմված վերնագրի ֆայլերի ցուցակներ սերվերի և բոտի համար:
CFLAGS-ը փոփոխական է, որը պարունակում է կոմպիլյատորների դրոշներ, ինչպիսիք են -Wall նախազգուշացումները միացնելու համար, -Error նախազգուշացումները որպես սխալ դիտարկելու համար և -std=c++98՝ C++98 ստանդարտը նշելու համար:
CC-ն փոփոխական է, որը նշում է C++ կոմպիլյատորը:
RM-ը փոփոխական է, որը նշում է ֆայլերը հեռացնելու հրամանը:
Makefile-ի թիրախներն են.

բոլորը. սա լռելյայն թիրախն է, որը կառուցում է հիմնական ծրագիրը ($(NAME)):
run_bot. Այս թիրախը կառուցում է բոտային ծրագիրը ($(BOT_NAME)):
objs և bot_objs. Այս թիրախները ստեղծում են թղթապանակներ՝ օբյեկտների ֆայլերը պահելու համար:
մաքուր. Այս թիրախը հեռացնում է օբյեկտի ֆայլերը և նախապես կազմված վերնագրերը:
fclean: Այս թիրախը կատարում է մաքրում և նաև հեռացնում հիմնական ծրագիրը:
re: Այս թիրախը վերակառուցում է ամբողջ ծրագիրը զրոյից:
թեստ. Այս թիրախը փորձարկման նպատակների համար է՝ տպելու օբյեկտների ֆայլերի ցուցակները և նախապես կազմված վերնագրերը:

Makefile-ի կանոնները սահմանում են, թե ինչպես հավաքել սկզբնաղբյուր ֆայլերը օբյեկտային ֆայլերի մեջ և ինչպես դրանք կապել վերջնական գործարկվող ծրագրերը ստեղծելու համար ($(NAME) և $(BOT_NAME)):

1.2 Սերվերային հատվածը և նրա բացատրությունը

Սերվերը պատասխանատու է հաճախորդի և մյուս սերվերների միջև կապ հաստատելու համար (Օր. Bot)
և այն պարունակում է մեկ հիմնական սերվեր և հինգ միջանկյալ ծրագրեր, ինչպիսիք են ալիքները և դրա վերահսկիչը, օգտվողները և դրա ձևերը, հրամանների կատարողն ու վավերացնողը, բացառությունների մշակիչը, հաղորդագրությունների կարգավորիչը և բոտի միացման կարգավորիչը:

1.2.1 հիմնական սերվեր և իր հիմնական աշխատանքը

սերվերը պատրաստել է singleton design pattern-ով
դա նշանակում է, որ ծրագրի սկզբից այն կհատկացվի միայն մեկ անգամ և մինչև գործընթացի ավարտը սերվերը կմնա նույն հիշողության հատվածում:
Սերվերի հիմնական մասը վարդակից կապի բացումն է որպես հոսք, որը կարող է բոլորը միանալ սերվերին ip-address-ի և դրա պորտի միջոցով:
վարդակների միացումները բացելու որոշ ֆունկցիոնալություն կա, որը նույնն է ծրագրավորման բոլոր լեզուների համար, և դրանք գրվել են 1965 թվականին, և դրանք երբեք չեն փոխվի:
կոդավորման կողմից սերվերը կոդավորված է երկու ֆայլում, մեկը՝ server.hpp և server.cpp
.hpp ֆայլերը ներառում են օգտագործված դասերի և գործառույթների նախատիպը, և .cpp-ն ներքևում գտնվող կոդի ներդրման մասն է, ես բացատրել եմ բոլոր գործառույթներն ու ալգորիթմները:

Server.hpp

```
#if !defined(SERVER_HPP)
#define SERVER_HPP

#include "ClientManager.hpp"
#include "Channel.hpp"
#include "CommandResponse.hpp"
```

```
class Server : public CommandResponse // Այս դասը ժառանգում է CommandResponse դասից
{
    private: // Սինգլտոնի համար
        static Server *instance; // Սինգլտոնի մեթոդը, որը հանգուցցները վերահաշվելու համար օգտագործվում է

    private:
        int master_socket; // Կայքագործիր սերվերի առաջարկով, կամ կատարեցիր ունիվերսալ ալարմ
        int max_sd; // Գծային բաշխվածության առավելագույն թույլատրականը
        int port; // Սերվերի պորտը
        std::string password; // Գաղտնաբառը
        int bot_fd; // Բոտի ֆայլի նշանակում
        std::map<std::string, Channel> channels; // Պահում է ալարմերը և նրանց արժեքները
        struct sockaddr_in address; // Սերվերի սոցիալական հասցե
        socklen_t   addrlen;
        fd_set readfds;

    public:
        Server(int port, std::string password);// Սերվերի կոնստրուկտորը
        ~Server();// Սերվերի ամփոփում

        static Server *getServer();// Ինչպես հետադարձ սերվեր հանգուցց ստանալու համար
        void Setup();// Սերվերի ստեղծում
        void ResetSockets();// Սոկետների վերակայում
        void CreateServer();// Սերվերի ստեղծում
        void SetOptions();// Կոնֆիգուրացնում սերվերը ասինխրոնական ռեժիմով
        void BindSocket();// Կցում սոկետը
        void StartListening();// Սկսում լսումը
        int AcceptNewSocket();// Սերվերի ավելացում
        void ListenForClientInput();// Մասնակցի գրանցում
        void SendToClient(int sockfd, const char *message) const;// Ուղարկում նամակ մասչելին
        void SendMessageToBot(const std::string &message) const;// Ուղարկում նամակ բոտին
        void WaitForActivity();// Սպասում գործարկումից
        void HandleIncomingConnections();// Բացական կապերի գործարկում
        void ClearClientFromChannels(const Client &client);// Սաքրում հաճախումներից
        int getaddrlen();// Ստանում ենթադրված երկարությունը
        struct sockaddr_in *GetAddress();// Ստանում հասցեն
        std::string const &getPass() const;// Ստանում գաղտնաբառը
```

```
    public:
        int getBotDescriptor() const;// Բոտի նշանակում
        void SetBotDescriptor(int new_fd);// Կարգավորում բոտը
        void RemoveBot();// Բոտի հեռացում
        bool IsBotConnected() const;// Ստուգում բոտը կապված է թե ոչ
        bool IsBot(const Client &client) const;// Ինչպես բոտը ստուգում

        bool HasChannel(std::string const &name);// Ունի ալարմ
        Channel &getChannel(std::string const &name);// Ալարմ
        std::string const getHost() const;// Հաճախումը

        void removeChannel(std::string const &name);// Ալարմի հեռացում
        void SendHelloMessage(const Client &client) const;// Ուղարկում
մատչելիին
};

#endif
```

Server.cpp

```
#include "Server.hpp"
#include "MessageController.hpp"

Server *Server::instance = NULL;

Server::Server()// Սերվերի դիտարկիչի կոնստրուկտոր
{

}

Server::Server(int _port, std::string _password) // Սերվերի ամփոփում
    : port(_port), password(_password), bot_fd(0)
{
    if (!instance)
        instance = this;
    else
    {
        std::cout << "Creating second instance of Server!!!" << std::endl
```

```
            << "Bad idea, try new tricks!!!" << std::endl;
        this->~Server();
    }
}

Server::~Server()
{

}

void    Server::Setup()
{
    addrlen = sizeof(address);
    CreateServer();
    SetOptions();
    BindSocket();
    StartListening();
}

void    Server::ResetSockets()
{
    FD_ZERO(&readfds);
    FD_SET(master_socket, &readfds);

    int max_fd_in_clients = ClientManager::getManager()-
>AddClientstToReadFds(&readfds);
    max_sd = std::max(master_socket, max_fd_in_clients);
}

void    Server::CreateServer()
{
    if( (master_socket = socket(AF_INET , SOCK_STREAM , 0)) == 0)
    {
        perror("socket failed");
        exit(EXIT_FAILURE);
    }
}

void Server::SetOptions()
{
    int opt = 1;
    if( setsockopt(master_socket, SOL_SOCKET, SO_REUSEADDR, (char *)&opt,
        sizeof(opt)) < 0 )
    {
        perror("setsockopt");
        exit(EXIT_FAILURE);
    }
```

```
}

void    Server::BindSocket()
{
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(port);
    if (bind(master_socket, (struct sockaddr *)&address, addrlen)<0)
    {
        perror("bind failed");
        exit(EXIT_FAILURE);
    }
}

void    Server::StartListening()
{
    std::cout << "Listener on port " << port << std::endl;
    // specifying maximum of 3 pending connections for the master socket
    if (listen(master_socket, 100) < 0)
    {
        perror("listen");
        exit(EXIT_FAILURE);
    }
    fcntl(this->master_socket,F_SETFL,O_NONBLOCK);
    std::cout << "Waiting for connections ..." << std::endl;
}

void    Server::ListenForClientInput()
{
    ClientManager::getManager()->HandleInput(&readfds);
}

void    Server::SendToClient(int sockfd, const char *message) const
{
    if (send(sockfd, message, strlen(message), 0) < 0)
        perror("send");
}

void    Server::SendMessageToBot(const std::string &message) const
{
    if (send(bot_fd, message.c_str(), message.length() + 1, 0) < 0)
        perror("send");
}

void    Server::WaitForActivity()
{
    int activity = select(max_sd + 1 , &readfds , NULL , NULL , NULL);
```

```
    if ((activity < 0) && (errno!=EINTR))
        std::cout << "select error" << std::endl;
}

int Server::AcceptNewSocket()
{
    int new_socket;
    if ((new_socket = accept(master_socket,
                (struct sockaddr *)&address, &addrlen))<0)
    {
        perror("accept");
        exit(EXIT_FAILURE);
    }
    return (new_socket);
}

void    Server::HandleIncomingConnections()
{
    int new_socket;
    if (FD_ISSET(master_socket, &readfds))
    {
        new_socket = AcceptNewSocket();
        //inform user of socket number - used in send and receive commands
        std::cout << "New connection , socket fd is " << new_socket
            << ", ip is : " << getHost() << ", port : "
            << ntohs(address.sin_port) << std::endl;

        ClientManager::getManager()->AddClient(new_socket);
    }
}

void    Server::ClearClientFromChannels(const Client &client)
{
    int socket = client.getSocket();
    for(std::map<std::string, Channel>::iterator it = channels.begin();
        it != channels.end(); it++)
    {
        it->second.LeaveIfMember(socket);
    }
}

Server  *Server::getServer()
{
    if (!instance)
        instance = new Server();
    return (instance);
```

```
}

int Server::getaddrlen() { return (addrlen); }

struct  sockaddr_in *Server::GetAddress()
{
    return (&address);
}

std::string const &Server::getPass()const
{
    return this->password;
}



bool Server::HasChannel(std::string const &name)
{
    std::map<std::string,Channel >::iterator it = channels.find(name);
    if(it != channels.end())
        return true;
    return false;
}

Channel &Server::getChannel(std::string const &name)
{
    if(!HasChannel(name))
        channels.insert(std::pair<std::string, Channel>(name, Channel(name)));
    return channels[name];
}

std::string const Server::getHost() const
{
    return (inet_ntoa(address.sin_addr));
}

void Server::removeChannel(std::string const &name)
{
    std::cout << "Removing Channel" << std::endl << std::endl;
    channels.erase(name);
}

void    Server::SendHelloMessage(const Client &client) const
{
    std::string mess = client.GetFormattedText() + " 001 " + client.getNick()
+ " :Welcome to irc server";
    SendMessageWithSocket(client.getSocket(), mess);
}
```

```
int     Server::getBotDescriptor() const
{
    return this->bot_fd;
}


void    Server::SetBotDescriptor(int new_fd)
{
    this->bot_fd = new_fd;
}

void    Server::RemoveBot()
{
    this->bot_fd = 0;
}

bool    Server::IsBotConnected() const
{
    return (bot_fd != 0);
}

bool    Server::IsBot(const Client &client) const
{
    return (bot_fd == client.getSocket());
}
```

1.2.1 Հիմնական սերվեր և իր հիմնական աշխատանքը

Channel.hpp

```
#if !defined(CHANNEL_HPP)
#define CHANNEL_HPP

#include "Client.hpp"
#include <vector>
#include <map>
#include "CommandResponse.hpp"


struct ModeType
{
    enum Mode
    {
```

```
        none = 0,
        read = 1,
        write_ = 2,
        invite = 4,
        private_ = 8
    };
};

class CommandResponse;

class Channel : public  CommandResponse
{
    public:

    // Կոնստրուկտորներ
    Channel();
    ~Channel();
    Channel(std::string const &_name);

    // Ադմինիստրատորները
    void MakeAdmin(int admin, int newAdmin);
    void RemoveFromAdmins(int admin, int oldAdmin);

    // Լրացուցիչ մեթոդներ
    void ChannelWhoResponse(Client const &client);
    void ChannelJoinResponse(Client const &client);
    void ChangeChannelUser(Client const &client);

    // Փոփոխումներ
    void SetPassword(const std::string &_password);

    // Հեռացումներ
    void RemoveMember(int admin, int removingMember);
    void LeaveMember(int memberNick);
    void LeaveIfMember(int memberNick);
    void KickMember(int admin, int removingMember);

    // Ադմինիստրատորի անունը և համարը
    int GetAdmin();
    std::string GetNickWithSocket(int socket) const;

    // Բոլոր մեթոդները և ինֆորմացիայի արժեքները
    void PrintData();
    std::string ModeInfo() const;
    int HasMode(ModeType::Mode _mode) const;
    int getMemberCount();
```

```
    // Վերահաշվարկներ
    bool IsAdmin(int memberNick) const;
    bool HasMember(int memberName) const;
    bool CheckPassword(const std::string &_checkingPass) const;

        int     HasMode(ModeType::Mode _mode)const;
        void    AddMode(ModeType::Mode mode);
        void    RemoveMode(ModeType::Mode mode);
        std::string ModeInfo() const;

        int     GetAdmin();
        std::string GetNickWithSocket(int socket) const;

        void ChannelWhoResponse(Client const &client);
        void ChannelJoinResponse(Client const &client);
        void ChangeChannelUser(Client const &client);

    private:
        std::string name;
        std::string password;
        std::map<int, Client> members;
        int     mode;
        void    ValidateAdmin(int admin) const;
        void    ValidateAdminIsInChannel(int admin) const;

        void    ValidateClientIsInServer(int client) const;
        void    ValidateClientIsInChannel(int admin, int client) const;

        void    ValidateCanModifyAdmin(int admin, int newAdmin) const;


    private:
        void    SetAdmin(int newAdmin);
        void    DeleteAdmin(int removingAdmin);

        mutable std::vector<int> admins;
};




#endif // CHANNEL_HPP
```

ClientManager.hpp

```cpp
// Այս ֆայլը հետևյալ արժեքները կառուցում է հետևյալը:
// IRC սերվերի չանելումների հետ աշխատանքի ամբողջականության կառավարիչ

#if !defined(CLIENT_MANAGER_HPP)
#define CLIENT_MANAGER_HPP

#include "Client.hpp"
#include "CommandHandler.hpp"
#include "irc.hpp"

class MessageController;

class ClientManager
{
private:
    std::map<int, Client>   clientMap; // Քանակային աղյուսակ հաճախորդների համար
    mutable std::map<int, Client>::const_iterator it; // Վերագրվող իտերատոր
    char buffer[1025]; // Բուֆեր

private:
    static ClientManager *instance; // Հիմնական կառավարիչի դիտարկ
    MessageController   *messageController; // Սեսաջանի կառավարիչ

public:
    bool    HasClient(int clientSocket) const; // Արդյունքում սերվերում առկա է հաճախորդը
    bool    HasClient(const std::string &clientNick) const; // Արդյունքում սերվերում առկա է հաճախորդի նիկն
    int     GetClientSocket(const std::string &clientName) const; // Ստանալ հաճախորդի սոցիալական համարը
    void    AddClient(int socketFd); // Ավելացնել հաճախորդ
    void    RemoveClient(int socketFd); // Հեռացնել հաճախորդ
    void    RemoveClient(std::map<int, Client>::iterator &iter); // Հեռացնել հաճախորդը ցուցակից

    int     AddClientstToReadFds(fd_set *readfds); // Ավելացնել կարդալու ֆայլի սուլների համար
    void    CloseClient(int clientSocket, const std::string &reason); // Փակել հաճախորդը
    void    HandleInput(fd_set *readfds); // Կառավարել ներմուծումը
    void    HandleMessage(Client &client); // Կառավարել նամակը
```

```
    ClientManager(); // Կառավարիչի կոնստրուկտոր
    ~ClientManager(); // Կառավարիչի դեստրուկտոր
    static ClientManager *getManager(); // Ստանալ կառավարիչի օբյեկտը
    const Client &getClient(int clientSocket) const; // Ստանալ հաճախորդը
    const Client &getClient(const std::string &clientSocket) const; // Ստանալ
հաճախորդը
};

#endif // CLIENT_MANAGER_HPP
```

MessageController.hpp

```
#if !defined(MESSAGE_CONTROLLER_HPP)
#define MESSAGE_CONTROLLER_HPP

#include "CommandData.hpp"
#include "Client.hpp"
#include <cstring>
#include <sstream>
#include <map>
#include <vector>
/**
* @class MessageController
*
* Այս դասը ծառայում է որպես IRC սերվերի հավելվածում հաղորդագրությունների
մշակման կենտրոնական հանգույց: Այն ներառում է գործառույթները հետևյալի համար.
*
* - Վերլուծություն. բաժանում է հաճախորդներից ստացված հաղորդագրությունները
առանձին IRC հրամանների, որոնք ներկայացված են «CommandData» օբյեկտներով:
* - Վավերացում. Ապահովում է, որ ալիքի և մականվան ձևաչափերը համապատասխանում են
IRC արձանագրության բնութագրերին:
* - Հաղորդագրությունների վերակառուցում. մշակում է պոտենցիալ մասնատված
հաղորդագրությունները, որոնք ստացվել են կտորներով՝ դրանք հավաքելով ամբողջական
հաղորդագրությունների:
* - Տողերի մանիպուլյացիայի կոմունալ ծառայություններ. տրամադրում է օգնական
գործառույթներ հաղորդագրության մշակման մեջ օգտագործվող սովորական տողային
գործողությունների համար:
*
* «MessageController»-ը պահպանում է ներքին քարտեզ («chunksMap»)՝ հատուկ
հաճախորդի վարդակների հետ կապված հաղորդագրությունների հատվածներին հետևելու
համար: Սա թույլ է տալիս դասին վերականգնել ամբողջական հաղորդագրությունները,
նույնիսկ եթե դրանք հասնում են մի քանի մասերի:
*/
class MessageController
```

```
{
    /**
     * «MessageController»-ի կանխադրված կոնստրուկտոր:
     */
    MessageController();

    /**
     * Destructor «MessageController»-ի համար: Պատասխանատու է պատշաճ մաքրման
     * համար (անհրաժեշտության դեպքում):
     */
    ~MessageController();

public:
    /**
     * Վերլուծում է պոտենցիալ բազմաթիվ IRC հրամաններ պարունակող տողը, որոնք
     * առանձնացված են տողերի ընդմիջումներով (`\n`):
     * Յուրաքանչյուր հրաման արդյունահանվում և վերածվում է համապատասխան
     * «CommandData» օբյեկտի՝ ձևավորելով վեկտոր, որը ներկայացնում է վերլուծված
     * հրամանները:
     *
     * @param մուտքագրում վերլուծվող հաղորդագրությունը պարունակող տողը:
     * @return «CommandData» օբյեկտների վեկտոր, որը ներկայացնում է վերլուծված
     * հրամանները:
     */
    std::vector<CommandData> Parse(std::string &input) const;


    /**
     * Վերլուծում է մեկ IRC հրամանի տողը (առանց տողերի ընդմիջումների)
     * «CommandData» օբյեկտի մեջ: Սա սովորաբար օգտագործվում է ավելի մեծ
     * հաղորդագրությունից արդյունահանվող առանձին հրամանների մշակման համար:
     *
     * @param singleCommand Տող, որը պարունակում է վերլուծվող մեկ հրաման:
     * @return «CommandData» օբյեկտ, որը ներկայացնում է վերլուծված հրամանը:
     */
    CommandData ParseSingleCommand(const std::string &singleCommand) const;

    /**
     * Տպում է «CommandData» օբյեկտների վեկտորի բովանդակությունը ելքային հոսքի
     * վրա (սովորաբար կոնսոլը կամ գրանցամատյան ֆայլը) վրիպազերծման կամ գրանցման
     * նպատակով:
     *
     * @param data Տպվող «CommandData» օբյեկտների վեկտորը:
     */
    void PrintData(std::vector<CommandData> &data) const;
```

```
    /**
     * Վավերացնում է, թե արդյոք տվյալ տողը համապատասխանում է վավեր IRC ալիքի
     անվանման ձևաչափի պահանջներին:
     *
     * @param channelName Վավերացվող տողը:
     * @return ճիշտ է, եթե տողը վավեր ալիքի անուն է, հակառակ դեպքում՝ կեղծ:
     */
    bool IsValidChannelName(const std::string &channelName) const;

    /**
     * Վավերացնում է, թե արդյոք տվյալ տողը համապատասխանում է վավեր IRC
     մականվան ձևաչափի պահանջներին:
     *
     * @param Մականուն Վավերացվող տողը:
     * @return ճիշտ է, եթե տողը վավեր մականուն է, հակառակ դեպքում՝ կեղծ:
     */
    bool IsValidNickname(const std::string &nickname) const;


    /**
     * Ստուգում է, արդյոք տրված տողը սկսվում է նշված նիշերի հավաքածուից որևէ
     նշանով:
     * Սա կարող է օգտակար լինել հատուկ հրամանների նախածանցները բացահայտելու
     համար (օրինակ՝ «/»):
     *
     * @param str Ստուգվող տողը:
     * @param set Նիշերի հավաքածու, որոնց դեմ պետք է ստուգել:
     * @return ճիշտ է, եթե տողը սկսվում է բազմության գրանշանով, հակառակ
     դեպքում՝ կեղծ:
     */
    bool StringStartsWithFromSet(const std::string &str, const std::string
     &set) const;

    /**
     * Որոշում է, թե արդյոք ստացվել է հաղորդագրության հատված որոշակի հաճախորդի
     վարդակից:
     * Սա օգտագործվում է մասամբ ստացված հաղորդագրություններին հետևելու համար,
     որոնք կարող են մասնատված լինել ցանցի սահմանափակումների պատճառով:
     *
     * @param clientSocket Հաճախորդի վարդակից նույնացուցիչը:
     * @return ճիշտ է, եթե հաճախորդի համար կա հաղորդագրության հատված, հակառակ
     դեպքում՝ կեղծ:
     */
    bool ContainsChunk(int clientSocket) const;

    /**
```

```
      * Հաճախորդից ստացված հաղորդագրության կտոր է ավելացնում ներքին քարտեզին
(`chunksMap`), որը պահում է հաղորդագրությունների հատվածները հաճախորդների համար
                մինչև ամբողջական հաղորդագրությունը կառուցվի:
                                        *
             * @param clientSocket Հաճախորդի վարդակից նույնացուցիչը:
      * @param messageChunk Ստացված հաղորդագրության հատվածը պարունակող տողը:
                                        */
        void    AddChunk(int clientSocket, const std::string &messageChunk);
                       void    ClearChunk(int clientSocket);
                std::string ConstructFullMessage(int clientSocket);
                  std::string trim(std::string const &str)const;
         std::string GetModesString(const std::string &argument, char sign)
                                  const;
               int SignCount(const std::string &str, char sign) const;



                  static  MessageController   *getController();

                                private:
               std::map<int, std::vector<std::string> > chunksMap;
                       static MessageController *instance;
                                };




                     #endif // MESSAGE_CONTROLLER_HPP
```

BOT

```
#if !defined(BOT_HPP)
#define BOT_HPP

#include "../server/irc.hpp"
```

```
/*

AcceptSocket() - Հաստատում է սոկետի ստացողը:
SendReply() - Ուղարկում է պատասխանը:
GiveResponse() - Տալու է պատասխանը նամակի հրահանգում:
RunBot() - Կաշխատում է բոտը:
Bot() - Կառավարվում է կառավարվող բոտը:
~Bot() - Մահվում է բոտը:
SetUser() - Սահմանում է օգտագործողը:
SetNick() - Սահմանում է անվանումը:
AddToRecvMsg() - Ավելացնում է ստացված նամակները:
GetRecvMsg() - Ստանում է ստացված նամակները:
 */

class Bot
{
    private:
        socklen_t   addrlen;
        struct sockaddr_in address;
        int             port;
        int             socketfd;
        int             clientfd;
        std::string     host;
        std::string     pass;
        std::string     user;
        std::string     nick;
        char            buffer[1025];
        std::string     recvMessage;

        void    Setup();//- Նախապատրաստում է սերվերը:
        void    CreateServer();//- Ստեղծում է սերվերը:
        void    SetOptions();//- Կազմում է սուպերատրիկ կապ:
        void    BindSocket();//- Հաստատում է սոկետը:
        void    ConnectToServer();//- Կապվում է սերվերին:
        void    ReceiveMsg();//- Ստանում է նամակներ:
        void    AcceptSocket();//- Հաստատում է սոկետի ստացողը:
        void    SendReply();//- Ուղարկում է պատասխանը:
        std::string GiveResponse(const std::string &command);//- Տալու է
պատասխանը նամակի հրահանգում:


    public:
        int socketCLIENT;
        void    RunBot();//- Կաշխատում է բոտը:
        Bot(const std::string &host, int _port, const std::string &_pass,
```

```
            const std::string &_user = "havayi", const std::string &_nick =
"butul");//- Կառավարվում է կառավարվող բոտը:
        ~Bot();//- Սահվում է բոտը:

        void    SetUser(const std::string &_user);//- Սահմանում է օգտագործողը:
        void    SetNick(const std::string &_nick);//- Սահմանում է անվանումը:

        void        AddToRecvMsg(const std::string &msg);//- Ավելացնում է
ստացված նամակները:
        std::string GetRecvMsg(void) const;//- Ստանում է ստացված նամակները:
};


#endif // BOT_HPP
```