



УНИВЕРЗИТЕТ
У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ
НАУКА У НОВОМ САДУ



Бојан Куљић

РЕПЛИКАТОР ПОДАТАКА

ПРОЈЕКАТ

Основне академске студије

Нови Сад, 2024

Садржај

1. Увод	1
1.1. Кориснички захтеви, очекивања и циљ пројекта.....	1
1.2. Приказ дизајна пројекта	2
2. Опис коришћених технологија и алата	3
2.1. Windows Socket API (Winsock).....	3
2.2. Стандардне C/C++ библиотеке	4
2.3. Структуре података	5
2.4. Нити(Threads)	6
2.5. Критична секција (Critical Section)	7
3. Опис решења проблем	8
3.1. Почетна архитектура пројекта	8
3.2. Синхрони режим репликације података	8
3.3. Асинхрони режим репликације података	10
3.4. Имплементација клијента у коду.....	11
3.5. Опције првобитне верзије клијента	11
3.6. Додатне опције допуњене верзије клијента	13
3.7. Имплементација сервера у коду	14
4. Опис експерименталног дела пројекта.....	15
4.1. Изглед корисничког менија за оба режима рада	15
4.2. Приказ успостављања конекције између сервиса	15
4.3. Опис опција менија и тесни прикази конзоле	16
5. Закључак	18
5.1. Предлози за могућа усавршавања пројекта.....	18
Литература	19

1. Увод

Овај документ је направљен да детаљно опише, објасни и читаоца уведе у причу о самој проблематици, коришћеним алатима, реализованим решењима и о могућим унапређењима пројекта који носи назив „Репликатор података“.

Овај пројекат је првобитно био реализован за потребе предмета „Индустријски комуникациони протоколи“ који се слуша у четвртој години, односно у седмом семестру, на смеру Примењено софтверско инжењерство по новој акредитацији Факултета техничких наука у Новом Саду. У договору са уваженим асистентом извршене су додатне корекције и имплементиране нове функционалности како би овај пројекат могао да се искористи за потребе предмета „Пројекат“.

1.1. Кориснички захтеви, очекивања и циљ пројекта

Свесни смо да свет у коме живимо сваког дана постаје све модернији и дигитализованији и да брзина преноса података игра једну од кључних улога у многим апликацијама и сервисима. Поред брзине преноса података неопходно је обезбедити сигуран, поуздан и ефикасан начин за пренос и репликацију (копирање) података између различитих система како би се осигурала њихова доступност и отпорност на губитак података.

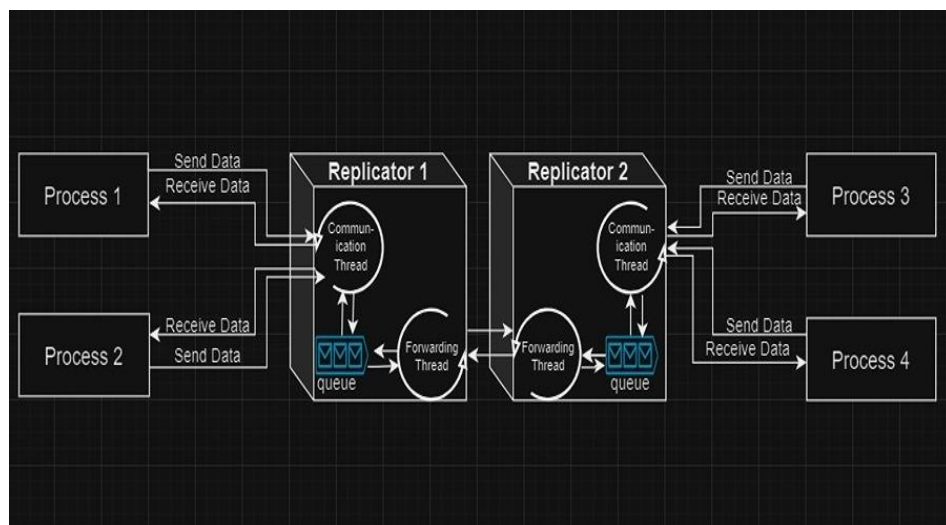
У складу са горе наведеним корисничким захтевима сам циљ нашег пројекта је имплементација сервиса за репликацију података између клијената. Може да постоји више клијената (процеса) и два репликатора (сервера) која су задужена за репликацију и комуникацију са клијентима. Наш програм подржава два начина преноса података: синхрони и асинхрони.

Од пројекта се очекује да има функционалну имплементацију метода за:

- Регистравање сервиса
- Слање порука
- Репликацију (за потребе асинхроног слања)
- Повратни позив (последња порука)
- Приказ свих порука
- Брисање порука (на основу прослеђеног броја)
- Статистику репликација
- Затварање конекције

1.2. Приказ дизајна пројекта

На слици 1.1 приказан је дизајн нашег репликатора података. Он служи за приказ илустрације одвијања и функционисања нашег програма, као и за приказ тока комуникације између различитих процеса и репликатора 1 и 2 помоћу нити (*threads*). Нити омогућују покретање више клијената (процеса) истовремено.



Слика 1.1. Приказ дизајна система за репликацију података

Репликатори садрже нити од којих је једна задужена за репликацију са суседним репликатором (*forwarding thread*), а остале за комуникацију са одговарајућим процесима, односно за слање и примање података. Поред тога у репликатору се налази листа у коју се увезују сви процеси коју се се регистровани на одређени сервис. Главна нит (*main thread*) је задужена за регистрацију нових корисника.

Главна компонента оба репликатора (сервера) је кружни бафер, на дијаграму означен са *queue*, у који се смештају пристигли подаци и даље прослеђују суседном репликатору. На овај начин је реализовано ефикасно коришћење меморије као и брз приступ подацима.

Кроз овај увод пружен је детаљан опис задатих почетних проблема, корисничких захтева и циљева. У наредним поглављима овог документа ће бити детаљно описане коришћене технологије и алати, решење проблема као и ток практичног дела пројекта уз методологију која је примењена током израде.

2. Опис коришћених технологија и алата

За развој апликације и имплементацију свих њених захтева и функционалности неопходно је модерно и високофункционално развојно окружење. За потребе пројекта кориштен је *Microsoft Visual Studio* са издањем *Community 2022* [1] и верзијом 17.10.1.

Кориштена платформа за развој софтвера је *.NET (Network Enabled Technologies) Framework* са верзијом 4.8.09037 [2]. Ова платформа подржава развој апликација које су „омогућене на мрежи“, што значи да могу да раде ефикасно и повезано са другим апликацијама преко мреже. У даљем тексту ће бити наведени кључни елементи, алати, библиотеке и технологије које су биле задужене за саму израду нашег програма.

2.1. Windows Socket API (Winsock)

Winsock (*Windows Socket*) API (*Application Programming Interface*) користи се за имплементацију мрежне комуникације у *Windows* окружењу и управљање сокетима [3]. Winsock омогућава апликацијама да комуницирају помоћу TCP (*Transmission Control Protocol*) или IP (*Internet Protocol*) протокола [4]. У табели 2.1 можемо да видимо списак неких функција и њихове особине које су кориштене током писања самог кода за успешну примену Winsock API-ја, успостављање конекције између клијента и сервера и размену података међу њима.

Назив функције	Особине и њена функционалност
<i>Socket</i>	Врши креирање нове утичнице (<i>socket-a</i>) на мрежи
<i>Bind</i>	Врши повезивање утичнице са неком локалном адресом
<i>Listen</i>	Додељује улогу утичници пасивног ослушаоца
<i>Accept</i>	Врши прихватање долазних података на утичници
<i>Recv & Send</i>	Врше примање и слање података
<i>Closesocket</i>	Извршава ослобађање ресурса и затварање конекције
<i>Ioctlsocket</i>	(<i>Input/Output Control Socket</i>) Користи неблокирајући режим

Табела 2.1. Називи и функционалности функција које су кориштене у имплементацији Winsock API-ја

2.2. Стандардне C/C++ библиотеке

Једна од највећих олакшица у програмирању јесте употреба библиотека. Библиотека је збирка унапред написаних кодова које програмери могу користити за обављање заједничких задатака без потребе да пишу код од нуле. Библиотеке пружају функције, класе и друге ресурсе који могу бити позвани у програму како би се олакшало програмирање и повећала продуктивност.

Како је наш пројекат писан у C и C++ програмском језику [5] кориштен је велики број стандардних библиотека које пружају низ улазно/излазних операција, управљање меморијом, манипулацију стринговима, бележење времена и многе друге опције. У табели 2.2 су наведене неке од најчешћих библиотека које су кориштене у имплементацији пројекта.

Библиотеке	Опис и функционалност
<i>Stdio.h</i>	Стандардна библиотека за улазно/излазне операције у C, користи се за функције попут <i>printf</i> и <i>scanf</i>
<i>Stdlib.h</i>	Нуди функције за управљање меморијом, конверзију типова и генерисање случајних бројева у C
<i>Conio.h</i>	Обезбеђује функције за рад са улазом и управљање конзолом
<i>Winsock2.h</i>	Пружа декларације и функције за коришћење <i>Windows Sockets</i> API-а за мрежну комуникацију
<i>Ws2tcpip.h</i>	Подржава новије TCP/IP протоколе и проширења, омогућавајући напредне мрежне функционалности
<i>Fstream</i>	C++ библиотека која је задужена за рад са датотекама
<i>Iostream</i>	Пружа функције за улазно/излазне операције у C++, омогућавајући лако читање и писање података у конзоли
<i>Chrono</i>	Пружа функционалности за рад са временом и датумима у C++, омогућавајући мерење времена и кашњења
<i>String</i>	Омогућава рад са стринговима у C++, пружајући алате за манипулацију текстуалним садржајем

Табела 2.2. Називи коришћених библиотека, њихов опис и функционалности

2.3. Структуре података

Помоћу структура података у нашем пројекту су представљени и реализовани наћини организације и складиштења података. Оне су нам омогућиле ефикасно управљање подацима и извршавање различитих операција над истим подацима [5].

У коду нашег пројекта су дефинисане чак 4 структуре података:

1. Структура *DATA* садржи низ карактера *data* дужине 100, која представља податак који се чува у кружном баферу. Јаснији изглед структуре приказан је сликом 2.2 приказ а).
2. Структура *circular_buffer* представља кружни бафер који садржи низ структура *DATA*, као и капацитет, индексе *head* и *tail* за упис и читање. Јаснији изглед структуре приказан је сликом 2.2 приказ б).

```
typedef struct data_st{
    char data[100];
} DATA;
```

а)

```
typedef struct {
    DATA* buffer;
    int capacity;
    int head;
    int tail;
} circular_buffer;
```

б)

Слика 2.2. Приказ а) структура података *data_st* и б) структура података за кружни бафер

3. Структура *PROCESS* представља процес у вашем систему са јединственим идентификатором *processId*, индексом *index* и прихваћеним сокетом *acceptedSocket*. Јаснији изглед структуре приказан је сликом 2.3 приказ в).
4. Структура *NODE_REPLICATOR* представља чвор у листи процеса репликатора. Садржи информације о процесу и показивач на следећи чвор. Јаснији изглед структуре приказан је сликом 2.3 приказ г).

```
typedef struct process_st {
    GUID processId;
    int index;
    SOCKET acceptedSocket;
} PROCESS;
```

а)

```
typedef struct node_st_replicator {
    PROCESS process;
    struct node_st_replicator* next;
} NODE_REPLICATOR;
```

б)

Слика 2.3. Приказ а) структура података *PROCESS* и б) структура података за *NODE_REPLICATOR*

2.4. Нити(*Threads*)

Основне јединице којима баратамо у пројекту и које нам омогућавају паралелно извршавање кода унутар процеса су нити [6]. Нити деле исти адресни простор процеса и могу паралелно приступати заједничким ресурсима, што омогућава ефикасније извршавање задатака и бољу искоришћеност ресурса система.

Као што смо и навели у уводу наш дизајн система се састоји од три кључне нити:

- **Main thread** представља главну нит која је задужена за регистровање и успостављања конекције између нових клијената.
- **Forwarding thread** је кључна нит која повезује два сервера репликатора и врши размену података сачуваних у њиховим кружним баферима.
- **Communication thread** је нит која нам омогућује истовремено покретање више клијената и њихово паралелно повезивање са репликаторима 1 или 2. Омогућује истовремену обраду више клијентских захтева.

За управљање и синхронизацију нитима у коду коришћене су различите функције:

- **CreateThread** је функција која је задужена за крирање нове нити унутар процеса и као повратну вредност добијамо *HANDLE* нове нити. *Handle* представља тип дескриптора који управља ресурсима на нашем рачунару. Најјаснији преглед ове функције као и списак свих атрибута које добијамо кроз њену повратну вредност видимо у листингу 2.1.

```
HANDLE CreateThread(
    LPSECURITY_ATTRIBUTES lpThreadAttributes,
    SIZE_T dwStackSize,
    LPTHREAD_START_ROUTINE lpStartAddress,
    LPVOID lpParameter,
    DWORD dwCreationFlags,
    LPDWORD lpThreadId
);
//Конкретан пример позива ове функције из кода
handleConnect= CreateThread(NULL, 0, &handleConnectSocket,
&connectSocket, 0, &funId);
```

Листинг 2.1. Приказ функције за крирање нити, повратне вредности и конкретан пример позива

- **WaitForSingleObject** омогућава главној нити да сачека док новокреирана нит не заврши извршавање.
- **CloseHandle** затвара *HANDLE* нити и ослобађа ресурсе.

2.5. Критична секција (*Critical Section*)

Критичне секције (*Critical Sections*) су механизам за синхронизацију у програмима са више нити, који омогућава да само једна нит приступа заједничком ресурсу у датом тренутку. Ово спречава условне трке и обезбеђује интегритет података. Иницијализују се и управљају помоћу функција:

- *InitializeCriticalSectionAndSpinCount*,
- *EnterCriticalSection*,
- *LeaveCriticalSection*
- *DeleteCriticalSection*

Најбољи начин примене односно имплементације критичних секција у коду можемо да видимо у листигу 2.2. Код имплементира функцију „cb_init“, која иницијализује кружни бафер и користи критичну секцију да осигура сигуран приступ у окружењу са више нити

```
...
// Дефиниција критичне секције
CRITICAL_SECTION csProcess;

void cb_init(circular_buffer* cb) {
    //Иницијализација критичне секције са спин бројачем
    InitializeCriticalSectionAndSpinCount(&csProcess, 80000400);

    //Улазак у критичну секцију
    EnterCriticalSection(&csProcess);
    cb->buffer = (DATA*)malloc(BUFFER_SIZE * sizeof(DATA));
    if (cb->buffer == NULL) {
        //Излазак из критичне секције
        LeaveCriticalSection(&csProcess);
        perror("Memory allocation failed");
        exit(EXIT_FAILURE);
    }
    cb->capacity = BUFFER_SIZE;
    cb->head = 0;
    cb->tail = 0;

    //Излазак из критичне секције
    LeaveCriticalSection(&csProcess);
}
```

Листинг 2.2. Приказ примене критичних секција у коду конкретно на функцији cb_init

3. Опис решења проблем

Сваки задатак има почетне услове и проблематику која треба да се реши и на што бољи начин имплементира у коду како би све опције биле функционалне и изводљиве. Захтеви и спецификација за наш пројекат су објашњени и изложени у уводном дјелу, а у овом паглављу ћу објаснити како је извршена реализација свих захтева и функционалности.

3.1. Почетна архитектура пројекта

Једна од битних ставки у имплементацији је архитектура самог система. Наш пројекат за репликацију података у оквиру простора за развој решења (*solution*) има имплементирана четири кључна пројекта која су међусобно увезана и сваки за себе одрађује дефинисане функционалности. Функције унутар пројеката су раздвојене у: документ са заглављима (*header*), где су само дефинисане све функције, и у главни извршни документ (*cpp*) у којем је извршена реализација дефинисаних функција из првог документа. Додатни пројекти који представљају саставне компоненте нашег главног пројекта су:

- *Common* који представља једну библиотеку унутар које се чувају сви заједнички елементи унутар пројекта.
- *Process* врши реализацију клијента и свих његових метода.
- *Replicator1* реализује сервис број 1 за прихватање података од клијената.
- *Replicator2* сервис који има сличну улогу као и *replicator1*.

Помоћу ових додатних пројеката и унутар њих се врши имплементација система за репликацију података са клијент-сервер комуникацијом која подржава синхрони и асинхрони као и вишепроцесорски начин рада.

3.2. Синхрони режим репликације података

Синхрони режим репликације података између сервера подразумева обраду података у реалном времену са непосредном потврдом о извршењу задатка. Ово је битно за одржавање конзистентности података и координацију између различитих компоненти система. Синхрони режим репликације у коду се реализује у функцији „*TestSync*“, која има више задужења:

- Врши проверу да ли је сервис регистрован
- Креира копије тренутних бафера

- Врши обраду, слање и испис свих података из тренутних бафера као резултат позива тест методе, наравно у синхронном режиму рада

Све ово функционалности које обавља „*TestSync*“ метода најбоље можемо да уочимо приказом листинга 3.3.

```
bool TestSync(int ServiceID) {
// Провера да ли је сервер регистрован
if (ServiceID == 0) {
    printf("Service not registered\n");
    return true;
}
// Креирање копије тренутних бафера
char tempSyncBuffer[BUFFER_SIZE];
strcpy_s(tempSyncBuffer, BUFFER_SIZE, syncMessageBuffer);
bool isSyncBufferEmpty = (strlen(syncMessageBuffer) == 0);

//Обрада података за слање
char messageToSend[BUFFER_SIZE];
snprintf(messageToSend, BUFFER_SIZE, "RESULTS TEST(all last
messages):\n\t\t %s ",
    isSyncBufferEmpty ? "Buffer is empty" : syncMessageBuffer);

// Слање података
int iResult = send(clientSocket, messageToSend,
(int)strlen(messageToSend) + 1, 0);
if (iResult == SOCKET_ERROR) {
    printf("send failed with error: %d\n", WSAGetLastError());
    closesocket(clientSocket);
    WSACleanup();
    return false;
}
// Испис података
printf("\n\nRESULTS TEST(all last messages):\n");
if (isSyncBufferEmpty) {
    printf("Synchronous buffer is empty.\n");
}else{
    char* token;
    int msgCount = 1;
    token = strtok(tempSyncBuffer, ";");
    while (token != NULL) {
        printf("%d. %s\n", msgCount, token);
        token = strtok(NULL, ";");
        msgCount++;
    }
}
return true;
}
```

Листинг 3.3. Приказ тест методе за синхрону репликацију и испис

3.3. Асинхрони режим репликације података

Асинхрони режим репликације података омогућава обраду података без чекања на непосредну потврду о извршењу задатка. Овај приступ смо имплементирали зато што је користан за смањење кашњења и повећање перформанси система, посебно у ситуацијама где је брзина обраде важнија од конзистентности података у реалном времену.

Имплементација овог режима репликације података у коду је реализована тако да се подаци испишу на други сервер тек када се позове метода за репликацију која је у коду реализована кроз функцију *“ReplicateHandler”*. Детаљан изглед ове функције, као и приказ делова кода који бележе почетно и завршно време репликације, које нам игра главну улогу за израчунавање статистике репликација и логовање свих тих података у датотеци је детаљно приказано у листингу 3.4.

```
bool ReplicateHandler(int ServiceID) {
    if (ServiceID == 0) {
        printf("Service not registered\n");
        return true;
    }
    char input[10];
    char data[BUFFER_SIZE] = "Replicate!@#$$%^&*";

    //време када је репликација започета
    auto start = std::chrono::high_resolution_clock::now();
    if (!ReceiveData(data, strlen(data))) {
        gets_s(input, sizeof(input));
        return false;
    }

    //време када је репликација завршена
    auto end = std::chrono::high_resolution_clock::now();
    std::chrono::duration<double, std::milli> replicationTime = end -
start;

    LogReplication(data, replicationTime.count(), serviceType);
    return true;
}
```

Листинг 3.4. Приказ имплементације методе за репликацију података у коду

У овом режиму, клијент шаље податке серверу који их памти у свом асинхронном баферу док не добије поруку од клијента *„Replicate!“* која сигнализира серверу и потврђује да треба извршити репликацију. Репликатор затим шаље све поруке другом репликатору и бележи их у датотеци.

3.4. Имплементација клијента у коду

Како нам је у спецификацији пројекта задато да апликација може да ради са више клијената истовремено, тај захтев нам је представљао најозбиљнији корак у имплементацији решења. Применом паралелног програмирања и употребом нити наша апликација је оспособљена да подржава истовремени рад са више покренутих процеса.

Клијент представља једну конзолну апликацију која користи *Windows Sockets* библиотеку за комуникацију са сервером. При покретању конзоле клијент бира да ли жели да пренос података врши у синхронном (опција 1) или асинхронном режиму (опција 2). Кроз имплементацију кода за конзолног клијента кориштена је валидација и додатни услови за проверу исправности унесених података као и исписи и упозорења у случају грешки проузрокованих погрешно унетом вредношћу. У следећим подпоглављима биће наведене и објашњене опције односно функционалности које клијент поседује.

Како је овај пројекат надограђен за потребе новог предмета „Пројекат“ навешћемо методе које су реализоване пре у склопу иницијалних захтева као и методе које су реализоване после како додатне опције за надоградњу.

3.5. Опције првобитне верзије клијента

Првобитна верзија је имала четири односно пет функционалности (пета је додата за потребе асинхроне репликације). Те функционалности су биле задужене, а и у надограђеној верзији имају исту улогу, за:

1. Регистравање сервиса
2. Слање порука
3. Репликацију (додатна опција за потребе асинхроног слања)
4. Повратни позив (последња порука)
5. Затварање конекције

Имплементација ових функционалности у коду извршена је креирањем једне *while(true)* петље која ће да пролази бесконачно пута кроз све ове опције док корисник коначно не унесе неку од исправних опција. Затим ће да се позове одређени *“Handler”* (обрађивач захтева) у складу са унесеном вредношћу, проследиће се предефинисани параметри и функцији вратити вредност 1 као потврда да је нека опција изабрана.

Јасан приказ имплементације и позива одређених метода у коду можемо да видимо на листингу 3.5 .

```
while (true)
{
    InitializeWindowsSockets();
    memset((char*)&serverAddress, 0, sizeof(serverAddress));
    printf("Select one of the options: \n");
    printf("    1.Register service \n");
    printf("    2.Send data \n");
    printf("    3.Replicate \n");
    printf("    4.Callback \n");
    printf("    5.Close \n");

    char input[DEFAULT_BUFLen];
    int option = 0;
    if (gets_s(input, sizeof(input)) != nullptr)
        option = atoi(input);
    fflush(stdin);
    if (option == 1) {
        serviceID = RegistrationHandler(serviceID,
serviceType);
        if (serviceID == -1) return 1;
    }
    else if (option == 2) {
        if (!SendDataHandler(serviceID, serviceType))
            return 1;
    }
    else if (option == 3) {
        if (!ReplicateHandler(serviceID)) return 1;
    }
    else if (option == 4) {
        if (!CallbackHandler(serviceID)) return 1;
    }
    else if (option == 5) {
        if (CloseHandler(serviceID))
            return 0;
        else
            return 1; }
}
```

Листинг 3.5. Приказ дела кода првобитне имплементације корисничког менија конзолног клијента

3.6. Додатне опције допуњене верзије клијента

Надоградња почетног пројекта захтева имплементацију три додатне методе у корисничком менију. Те методе се налазе у менију под редним бројевима 5, 6 и 7 док је метода за затварање конекције пребачена под редни број 8. Додатне методе су задужене за:

- Приказ свих порука
- Брисање порука (на основу прослеђеног броја)
- Статистику синхроне и асинхроне репликације

У листингу 3.6 видимо надограђу почетне верзије кода.

```
...
printf("    5.Test(show data) \n");
printf("    6.Delete messages \n");
printf("    7.Replication statistics(ASYNC)\n");
...
else if (option == 5) {
    if (!TestHandler(serviceID, serviceType)) return 1;
}
else if (option == 6) {
    printf("Enter the number of last messages to delete: ");
    int messageCount = 0;
    if (gets_s(input, sizeof(input)) != nullptr)
        messageCount = atoi(input);
    fflush(stdin);

    if (messageCount == 0) {
        printf("Can`t deleted 0 messages!\n");
    }
    else {
        if (!DeleteMessage(serviceID, messageCount)) return 1;
    }
}
else if (option == 7) {
    ShowReplicationStatistics(serviceType);
}
else if (option == 8){
    if(CloseHandler(serviceId))
        return 0;
    else
        return 1;
}
```

Листинг 3.6. Приказ имплементације корисничког менија клијента са новим додатним методама

Имплементација **тест** методе за приказ свих порука није била захтевна за имплементацију. Подаци послати у одређеном режиму рада се чувају у засебним баферима. *TestHandler* метода провера који режим репликације је корисник изабрао и на основу тога позива *TestSync* или *TestAsync* методе за испис свих података из одређеног бафера на конзолу.

Метода за **брисање** прослеђеног броја порука је реализована у „*DeleteMessage*“ методи која такође проверава који режим рада је употребљен, копира податке из одређеног бафера у привремени бафер, врши бројање сачуваних порука унутар њега, упоређује прослеђени број са конзоле са тим бројем и ако постоји толико сачуваних података у тренутном складишту позива се метода „*free*“ за ослобађање односно брисање тог броја порука, у супротном се брише све.

Метода за приказ синхроне и асинхроне **статистике репликација** је била најзанимљивија за имплементацију. Захтевала је примену логовања порука у текстуалну датотеку заједно са њиховим временом потребним за репликацију. То време је рачунато тако што се одузму два забележена времена (када је репликатор примио поруку и када је клијент послао поруку). На основу ових података је реализована статистика о репликацијама која приказује: број репликација, најбрже, најлошије и просечно време потребно за репликацију као и њену брзину која се одређује на основу просечног времена за репликацију података и може бити: савршена, просечна и спора. Изглед и испис свих ових опција можете видети у следећем поглављу везаном за експериментални део пројекта.

3.7. Имплементација сервера у коду

Као што смо већ споменули наш пројекат има два сервиса-репликатора (1 и 2). Глави процес сервиса има улогу да прима нове клијенте тако што увек одржава отворену везу на слободном сокету и креира нову нит за интеракцију са клијентом. Када се одређени процеси конектују на сервис врши се обрада клијентских порука, њихово складиштење и репликација. На серверској страни је предефинисано 5 специјалних порука, да у случају ако оне стигну од клијента тада сервис мора за сваку од њих да изврши посебне активности. Те поруке су:

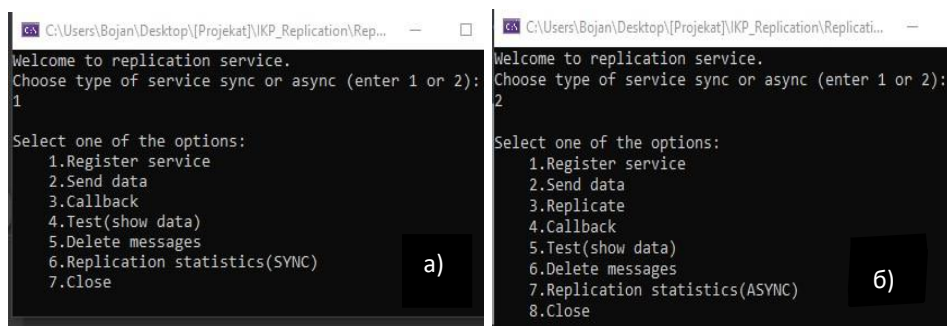
- „*CallBack!*“ враћа назад клијенту прослеђену поруку
- „*ReadSavedData!*“ чита све податке из датотеке и враћа их клијенту
- „*Replicate!*“ извршава репликацију података за асинхрон режим рада
- „*Close!*“ затвара комуникацију са репликатором и гаси тренутну нут
- „*Save!*“ врши бележење поруке у текстуалну датотеку

4. Опис експерименталног дела пројекта

Експериментални део пројекта обухвата тестирање синхроног и асинхроног режима репликације. Циљ је да се прикажу све функционалности, информације о успостављању комуникација, повратне поруке као и процене перформанси система у погледу брзине обраде захтева, поузданости репликације и укупне ефикасности управљања ресурсима.

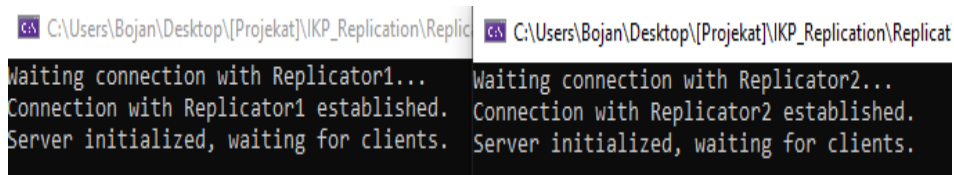
4.1. Изглед корисничког менија за оба режима рада

На слици 4.4 можемо да видимо изглед корисничке конзоле са исписом свих опција. Поглед а) представља опције за тип 1 - синхрона репликација, а поглед б) представља опције за тип 2 - асинхрона репликација.



4.2. Приказ успостављања конекције између сервиса

Са покретањем нашег пројекта и паралелним извршавањем два процеса за успоставу режима репликације покренута су и наша два сервис репликатора који прво проверавају статус сокета и исправност адреса за повезивање, а затим се међусобно повезују успостављајући стабилну конекцију и чекају регистрацију првог клијента. Тај приказ видимо на слици 4.5 .



4.3. Опис опција менија и тесни прикази конзоле

Први корак који мора да се изврши у тестирању функционалности јесте позив прве методе за регистрацију клијента на један од два понуђена сервиса.

Тесни случај: На репликатор 1 смо регистровали два клијента у различитим режимима рада (слика 4.7).

Помоћу опције 2 вршимо слање одређених порука и оне се сместају у дефинисано складиште.

Тесни случај: Послате две поруке у синхронном режиму рада са клијента 1 и једна порука са клијента 2 у асинхронном режиму (слика 4.7).

Трећа опција **код асинхроног клијента** је додатна и једина на основу које се разликују ова два менија. Позивом ове методе подаци са репликатора 1 ће се проследити и исписати на репликатор 2.

Опција 3 код првог и опција 4 код другог менија су задужене за испис последњих послатих порука од клијената на сервис (слика 4.7).

Опција 4 код првог и опција 5 код другог менија су задужене за позив тест методе и приказ свих примљених порука од клијената на сервис. Клијент 1 врши испис порука из синхроног бафера а клијент 2 из асинхроног бафера (слика 4.7).

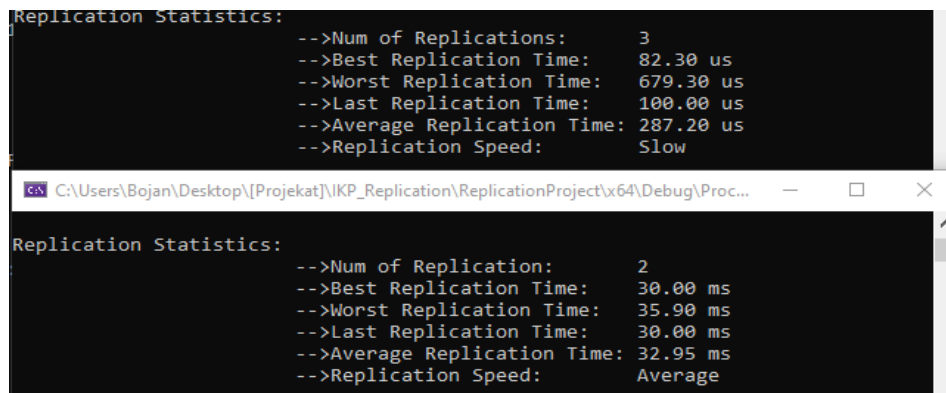
Опција 5 код првог и опција 6 код другог менија су задужене за брисање унесеног броја порука. У случају да се проследи 0 порука добићемо обавештење да није могуће брисање. Ако проследимо број порука за брисање који је већи од оног колико има порука у баферу, исписат ће се стање о тренутним порукама и извршити брисање целог бафера. Приказ ових опција видимо на листингу 4.7 .

```
6 (асинхрони мени)
Enter the number of last messages to delete: 0
Can't deleted 0 messages!

5 (синхрони мени)
Enter the number of last messages to delete: 3
Buffer contains only 2 messages.
Deleting all messages....
Successfully deleted 2 last messages.
```

Листинг 4.7. Пример извршавања функције за брисање прослеђеног броја порука код асинхроног и синхроног режима репликације података

Опција 6 код првог и опција 7 код другог менија врше приказ података о статистици за синхрону и асинхрону репликацију. Конкретан пример видите на слици 4.6.



```

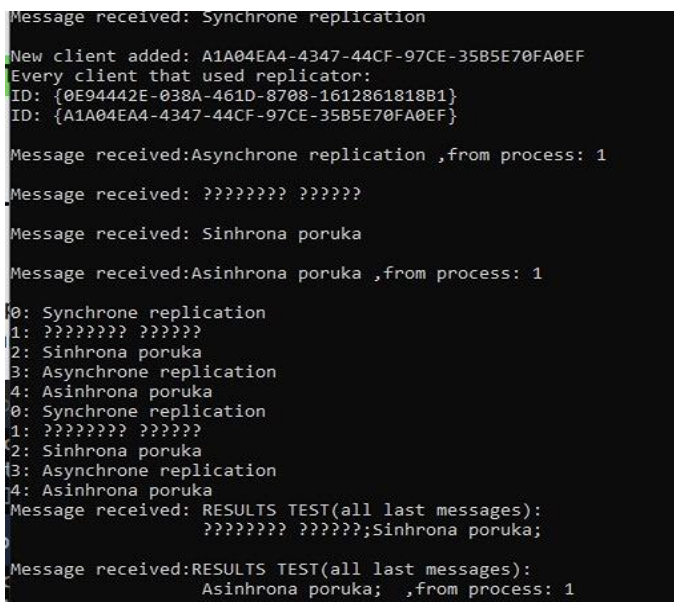
Replication Statistics:
-->Num of Replications:      3
-->Best Replication Time:    82.30 us
-->Worst Replication Time:   679.30 us
-->Last Replication Time:    100.00 us
-->Average Replication Time: 287.20 us
-->Replication Speed:       Slow

C:\Users\Bojan\Desktop\[Projekat]\IKP_Replication\ReplicationProject\x64\Debug\Proc...
Replication Statistics:
-->Num of Replication:      2
-->Best Replication Time:    30.00 ms
-->Worst Replication Time:   35.90 ms
-->Last Replication Time:    30.00 ms
-->Average Replication Time: 32.95 ms
-->Replication Speed:       Average
  
```

Слика 4.6. Приказ статистике о репликацији података

Опција 7 код првог и опција 8 код другог менија је задужена за затварање конекције са сервисима и гашење клијента.

Испис на репликатору 1 након позваних и истестираних свих метода можемо да видимо на слици 4.7.



```

Message received: Synchron replication
New client added: A1A04EA4-4347-44CF-97CE-35B5E70FA0EF
Every client that used replicator:
ID: {0E94442E-038A-461D-8708-1612861818B1}
ID: {A1A04EA4-4347-44CF-97CE-35B5E70FA0EF}

Message received:Asynchrone replication ,from process: 1
Message received: ???????? ???????
Message received: Sinhrona poruka
Message received:Asinhrona poruka ,from process: 1

0: Synchron replication
1: ???????? ???????
2: Sinhrona poruka
3: Asynchrone replication
4: Asinhrona poruka
0: Synchron replication
1: ???????? ???????
2: Sinhrona poruka
3: Asynchrone replication
4: Asinhrona poruka
Message received: RESULTS TEST(all last messages):
????????? ???????;Sinhrona poruka;
Message received:RESULTS TEST(all last messages):
Asinhrona poruka; ,from process: 1
  
```

Слика 4.7. Испис на конзоли репликатора 1

5. Закључак

Током писања ове пројектне документације прошли смо кроз сваки део пројекта, представили читаоцу првобитну проблематику и захтеве. Направили смо списак свих коришћених технологија и алата и описали имплементацију и решење самог пројекта у коду и све те функционалности приказали кроз експериментални део. Сада када смо дошли до завршног дела овог документа мислим да пројекат „Репликатор података“ има све неопходне функционалности и особине и да представља пример успешне реализације система за реплицирање података између сервиса.

Током рада били смо фокусирани на имплементацију система користећи два кључна приступа подацима помоћу којих се врши њихово слање и репликација, а то су синхрони и асинхрони приступ.

Кроз експериментални део, тестирали смо поузданост и перформансе система у различитим сценаријима, укључујући паралелну повезаност више клијената, обрађивање више клијентских захтева и управљање ресурсима. Резултати су показали да систем стабилно функционише и ефикасно користи ресурсе без неочекиваних радњи и испада, што је потврђено кроз праћење конзола.

5.1. Предлози за могућа усавршавања пројекта

Како ни један пројекат није савршен тако и наш оставља много простора за надоградњу и имплементацију нових функционалности које би побољшале, модернизовале и подигле сложеност овог пројекта на виши ниво.

Неки од предлога за будућа усавршења су :

- Имплементација „*thread pool-a*“ ради ефикаснијег управљања нитима приликом креирања нових процеса. Ово би могло смањити оптерећење система, убрзати одзив и боље искористити ресурсе.
- Оптимизација управљања порукама у кружном баферу. Тренутно поруке остају у баферу након што су исписане, па би увођење механизма за брисање порука након успешне репликације допринело ефикаснијем управљању меморијом и смањењу непотребног заузећа ресурса.
- Увођење механизма за компресију података пре њихове репликације. Овим механизмом би смањили количину пренетих података и убрзали процес репликације, посебно у мрежним окружењима са ограниченом пропусношћу.

Литература

[1] Microsoft Visual Studio 2022 Community Edition – Download Latest Version.

Доступно на:

<https://visualstudio.microsoft.com/downloads>.

[2] Microsoft .NET Framework 4.8 | Free official downloads.

Доступно на:

<https://dotnet.microsoft.com/en-us/download/dotnet-framework/net48>.

[3] Microsoft Windows Sockets 2 – Win32 apps. Доступно на:

<https://learn.microsoft.com/en-us/windows/win32/winsock/windows-sockets-start-page-2>.

[4] Brian Komar. “Teach Yourself TCP/IP Network Administration.”

Доступно за куповину на: <https://www.amazon.com/Teach-Yourself-TCP-Network-Administration/dp/0672312506>.

[5] Bjarne Stroustrup. “The C++ Programming Language.” Доступно на:

https://chenweixiang.github.io/docs/The_C++_Programming_Language_4th_Edition_Bjarne_Stroustrup.pdf.

[6] Steve Kleiman & Devang Shah. “Programming With Threads” First Edition.

Доступно за куповину на:

<https://www.amazon.com/Programming-Threads-Steve-Kleiman/dp/0131723898>.