

UNIVERSITETI I PRISHTINËS
FAKULTETI I INXHINIERISË ELEKTRIKE DHE KOMPJUTERIKE



Detyra: Implementim i IPC në Windows për
klient-server me teknikën message passing

Lënda: Sisteme Operative

Mentorët:

Prof. Asoc. Dr. Isak Shabani

Msc. Artan Mazrekaj

Kandidatët:

Agon Hoxha

Amire Gërguri

Përmbajtja

Përshkrimi i Problemit	2
Teknologjia	2
Implementimi	3
Zgjidhja 1 – Aplikacioni Server.....	3
Zgjidhja 1 – Aplikacioni Klient	4
Zgjidhja 1 – Shpjegim.....	5
Zgjidhja 2 – Komunikim ndërmjet threads me message passing.....	7
Zgjidhja 2 – Shpjegim.....	8
Referencat	9

Tabela e figurave

Figura 1. Klienti (majte) dhe Serveri (djathtë)	5
Figura 2. Disa klient të qasur në një server të vetëm	6
Figura 3. Rezultati i ekzekutimit.....	8

Përshkrimi i Problemit

Problemi i paraqitur nga detyra është mjaft i thjeshtë dhe lehtë i zgjidhshëm. Kërkesat janë që në sistemin operativ Microsoft Windows:

1. Të krijohet një aplikacion server.
2. Të krijohet një aplikacion klient.
3. Të mundësohet komunikimi mes këtyre proceseve.
4. IPC të jetë i llojit message passing.

Ndryshe nga disa detyra tjera që kanë qenë të shpërndara, për këtë detyrë nuk ka pasur shumë specifikime. Në fakt, tërë kërkesat janë të shprehura në një fjali të vetme. Për shkak të kësaj, e kemi pa si të arsyeshme që t'i ofrojmë dy zgjidhje për të shqyrtuar problemin.

Zgjidhja e parë e ofruar është komunikim mes dy proceseve – mes dy aplikacioneve të ndryshme. Zgjidhja e dytë është komunikim mes threadëve të ndryshëm brenda procesit të njëjtë. Të dyja, siq kërkuar, përdorin teknikën message passing.

Teknologjia

Për zgjidhjen e parë:

- Editori: Microsoft Visual Studio Community 2015
- Gjuha: C#
- Platforma: Windows 10

Për zgjidhjen e dytë:

- Editori: Eclipse IDE for Java Developers 2018-12
- Gjuha: Java
- Platforma: Windows 10

Implementimi

Zgjidhja 1 – Aplikacioni Server

```
using System;
using System.Runtime.Remoting.Channels.Ipc;
using System.Security.Permissions;

public class Server { //Shkruar nga: Agon Hoxha
    [SecurityPermission(SecurityAction.Demand)]
    public static void Main(string[] args){
        IpcChannel serverChannel=new IpcChannel("localhost:9090");//Krijo server channel
        // Regjistro channelin e serverit dhe percakto sigurine
        System.Runtime.Remoting.Channels.ChannelServices.RegisterChannel(serverChannel,
false);
        Console.WriteLine("Emri i channel eshte {0}.",serverChannel.ChannelName);
        // Trego emrin ^ dhe prioritetin e channel \
        Console.WriteLine("Prioriteti i channel eshte {0}.",
serverChannel.ChannelPriority);
        System.Runtime.Remoting.Channels.ChannelDataStore channelData =
            (System.Runtime.Remoting.Channels.ChannelDataStore)serverChannel.ChannelData;
        foreach (string uri in channelData.ChannelUris){
            Console.WriteLine("URI e channel eshte {0}.", uri);}
        // Lejo qasjen ne nje object per remote call
        System.Runtime.Remoting.RemotingConfiguration.
            RegisterWellKnownServiceType(typeof(RemoteObject), "RemoteObject.rem",
            System.Runtime.Remoting.WellKnownObjectMode.Singleton);
        // Parso URLne e channel
        string[] urls = serverChannel.GetUrlsForUri("RemoteObject.rem");
        if (urls.Length > 0){
            string objectUrl = urls[0], objectUri,
            channelUri = serverChannel.Parse(objectUrl, out objectUri);
            Console.WriteLine("URI e objektit eshte {0}.", objectUri);
            Console.WriteLine("URI e channel eshte {0}.", channelUri);
            Console.WriteLine("URL e objektit eshte {0}.", objectUrl);
        }
        Console.WriteLine("Per t'e ndalur serverin, shtyp 0 dhe ENTER.");
        // Prit per ndalim
        String uInput="";
        while (!uInput.Equals("exit")) {
            try{uInput = Console.ReadLine();}
            catch{uInput = "";}
        }
        if (uInput.Equals("exit")){
            serverChannel.StopListening(null);
            Console.WriteLine("Serveri eshte ndalur. Shtyp ENTER per t'e ndalur.");
            Console.ReadLine();
        }
    }
}

public class RemoteObject : MarshalByRefObject{
    private int callCount = 0;
    public int GetCount(){
        callCount++;
        Console.WriteLine("Metoda GetCount eshte thirrur " + callCount + " here.");
        return (callCount);
    }
}
```

Zgjidhja 1 – Aplikacioni Klient

```
using System;
using System.Runtime.Remoting.Channels.Ipc;
using System.Security.Permissions;

public class Client //Shkruar nga: Amire Gerguri
{
    [SecurityPermission(SecurityAction.Demand)] //wtf even is this
    public static void Main(string[] args)
    {
        // Krijo channel
        IpcChannel channel = new IpcChannel();
        // Regjistro channel, percakto sigurine
        System.Runtime.Remoting.Channels.ChannelServices.RegisterChannel(channel, false);
        // Regjistro se qka do dergohet, dhe ku
        System.Runtime.Remoting.WellKnownClientTypeEntry remoteType =
            new System.Runtime.Remoting.WellKnownClientTypeEntry(
                typeof(RemoteObject),
                "ipc://localhost:9090/RemoteObject.rem");
        System.Runtime.Remoting.RemotingConfiguration.
            RegisterWellKnownClientType(remoteType);
        // Krijo nje pool te mesazheve
        string objectUri;
        System.Runtime.Remoting.Messaging.IMessageSink messageSink =
            channel.CreateMessageSink(
                "ipc://localhost:9090/RemoteObject.rem", null,
                out objectUri);
        Console.WriteLine("URI e pool te mesazheve eshte {0}.",
            objectUri);
        if (messageSink != null){
            Console.WriteLine("Lloji i pool te mesazheve eshte {0}.",
                messageSink.GetType().ToString());
        }
        // Krijo instanc te RemoteObject
        RemoteObject service = new RemoteObject();
        // Thirr metoden e RemoteObject
        again:
        Console.WriteLine("Klienti po therret remote object.");
        Console.WriteLine("Remote objecti eshte thirrur "+service.GetCount()+" here.");
        // Mundesi e thjeshte per te perseritur thirrjen
        Console.Write("Deshironi t'e perseritni? (1=po, 0=jo): ");
        int repeat = 1;
        try{repeat = Int32.Parse(Console.ReadLine());}
        catch{repeat = 1;}
        if (repeat==1) goto again;
    }
}

public class RemoteObject : MarshalByRefObject
{
    private int callCount = 0;
    public int GetCount(){
        Console.WriteLine("GetCount has been called.");
        callCount++; return (callCount);
    }
}
```

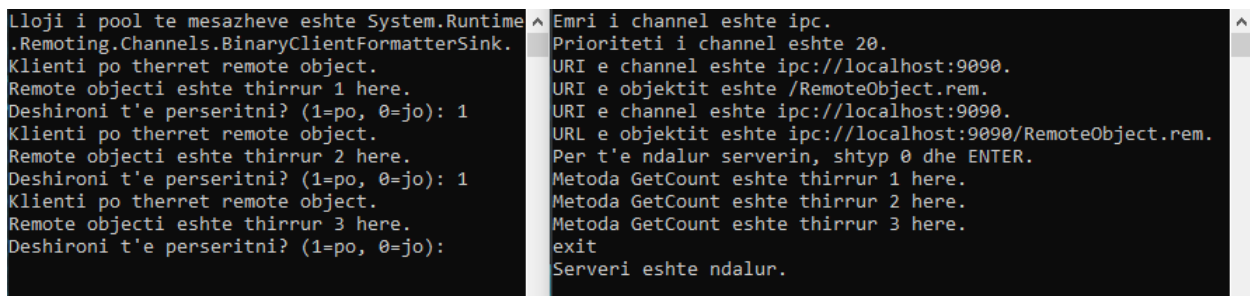
Zgjidhja 1 – Shpjegim

Zgjidhja e parë, e shkruar në C#, funksionon përmes IPC Channel. Thjeshtë thënë, kjo klas mundëson krijimin e një kanali të cilin dy programe mund t’ë përdorin për të komunikuar me njëra tjetrën. është një formë e thjeshtë e IPC komunikimit përmes message passing.

Pasi që kjo klasë krijon channel dhe mundëson komunikim pa ndonjë lloj ndalese të aplikuar, është e rekomanduar që të caktojmë permissions në kod tonë. Zakonisht përcakton vizibilitetin dhe qasjen që do kenë aplikacione që do lidhen me të, mirëpo në këtë rast është përdorur atributi i saj SecurityAction, që përcakton se qdo thirres duhet të ketë permission të njëjt. Mundëson parandalim të abuzimit të kodit. Nuk është gjithsesi e nevojshme, mirëpo është një rekomandim kur të përdoret një IPC Channel.

Kur klienti bën kërkesë, serveri lejon qasje në një objekt. Në këtë rast, lejohet qasja në të gjitha variablat dhe metodat e objektit. Nëse klienti ekzekuton një metod të objektit, do ekzekutohet tek serveri dhe do bartet rezultati i ekzekutimit. Kjo na kujton mënyrën se si funksionon PHP – ekzekutimi bëhet tek serveri dhe rezultati bartet tek klienti.

Në rastin tonë, e kemi marr një rast të thjeshtë. Kemi një objekt i cili ka një metod që inkrementon numrin se sa herë është thirrur objekti – pra, sa herë i është qasur klienti objektit të serverit. Numri se sa herë klienti është qasur në server, i kthehet klientit dhe shfaqet në dy anët në të njëjtën kohë – kjo thjeshtë të tregojmë se ka konsistenc dhe se të dy e shfaqin rezultatin e njëjtë se sa herë është thirrur metoda.



```
Lloji i pool te mesazheve eshte System.Runtime
.Remoting.Channels.BinaryClientFormatterSink.
Klienti po therrret remote object.
Remote objekti eshte thirrur 1 here.
Deshironi t'e perseritni? (1=po, 0=jo): 1
Klienti po therrret remote object.
Remote objekti eshte thirrur 2 here.
Deshironi t'e perseritni? (1=po, 0=jo): 1
Klienti po therrret remote object.
Remote objekti eshte thirrur 3 here.
Deshironi t'e perseritni? (1=po, 0=jo):

Emri i channel eshte ipc.
Prioriteti i channel eshte 20.
URI e channel eshte ipc://localhost:9090.
URI e objektit eshte /RemoteObject.rem.
URI e channel eshte ipc://localhost:9090.
URL e objektit eshte ipc://localhost:9090/RemoteObject.rem.
Per t'e ndalur serverin, shtyp 0 dhe ENTER.
Metoda GetCount eshte thirrur 1 here.
Metoda GetCount eshte thirrur 2 here.
Metoda GetCount eshte thirrur 3 here.
exit
Serveri eshte ndalur.
```

Figura 1. Klienti (majtë) dhe Serveri (djathtë)

Nëse ekzekutohet me sukses, dhe nuk ka ndonjë firewall që bllokon portin e përdorur, serveri sapo të fillon ekzekutimin shfaq të dhëna mbi vetveten, duke treguar emrin që e ka, prioritetin, URI të kanalit, URI të objektit, URL të kanalit, URL të objektit, si dhe jep një mesazh se si mund të ndalet serveri. Serveri do jetë gjithnjë në pritje që një klient t’i qaset, përpos nëse e ndalim. Ndalja e serverit bëhet përmes komandës exit.

Klienti, në anën tjetër, sapo të ekzekutohet, tregon llojin e pool të mesazheve, dhe tregon se është thirrur metoda në server. Gjithashtu jep mundësinë që të përsërit thirrjen sa do shumë herë që të dëshiroj përdoruesi. Duke shtypur 0, më nuk dërgohen thirrje tek objekti në server.

Serveri vetvetiu ruan gjendjen e numrit se sa herë klientët i janë qasur objektit për aq gjatë sa është në ekzekutim – këtu “klientët” është fjala kyqe që duhet vërejtur. Zgjidhja e ofruar nuk funksionon vetëm me një server dhe një klient, mirëpo ashtu sikur në botën reale, lejon që në një server të dërgojnë kërkesa një numër sado i madh i klientëve, dhe të gjithë mund të bëjnë thirrje në objektin e njëjtë në server, dhe të marrin mesazh se sa herë është thirrur metoda e objektit në server.

IPC Channel është një aplikim i veçant i teknikës message passing pasi që krijon një adresë (URL) nga e cila mund të marrim mesazhe, apo të dërgojmë në të – gjë që na ngjason me arkitektura të ndryshme në jetë reale si email.

Në vazhdim është shfaqur edhe rasti kur gjerja e kemi në ekzekutim serverin, i ekzekutojmë disa (3) klient dhe nga ta qasemi në objektin e njëjtë në server të njëjtë.

```

C:\Users\agoni\Desktop\OSP2C.exe
Lloji i pool te mesazheve eshte System.Runtime.Remoting.Channels.BinaryClientFormat
terSink.
Klienti po therret remote object.
Remote objecti eshte thirrur 3 here.
Deshironi t'e perseritni? (1-po, 0=jo):
Klienti po therret remote object.
Remote objecti eshte thirrur 5 here.
Deshironi t'e perseritni? (1-po, 0=jo):
Klienti po therret remote object.
Remote objecti eshte thirrur 8 here.
Deshironi t'e perseritni? (1-po, 0=jo):
Klienti po therret remote object.
Remote objecti eshte thirrur 11 here.
Deshironi t'e perseritni? (1-po, 0=jo):
Klienti po therret remote object.
Remote objecti eshte thirrur 14 here.
Deshironi t'e perseritni? (1-po, 0=jo):
Klienti po therret remote object.
Remote objecti eshte thirrur 16 here.
Deshironi t'e perseritni? (1-po, 0=jo):

C:\Users\agoni\Desktop\OSP2S.exe
Per t'e ndalur serverin, shtyp 0 dhe ENTER.
Metoda GetCount eshte thirrur 1 here.
Metoda GetCount eshte thirrur 2 here.
Metoda GetCount eshte thirrur 3 here.
Metoda GetCount eshte thirrur 4 here.
Metoda GetCount eshte thirrur 5 here.
Metoda GetCount eshte thirrur 6 here.
Metoda GetCount eshte thirrur 7 here.
Metoda GetCount eshte thirrur 8 here.
Metoda GetCount eshte thirrur 9 here.
Metoda GetCount eshte thirrur 10 here.
Metoda GetCount eshte thirrur 11 here.
Metoda GetCount eshte thirrur 12 here.
Metoda GetCount eshte thirrur 13 here.
Metoda GetCount eshte thirrur 14 here.
Metoda GetCount eshte thirrur 15 here.
Metoda GetCount eshte thirrur 16 here.
Metoda GetCount eshte thirrur 17 here.
Metoda GetCount eshte thirrur 18 here.

C:\Users\agoni\Desktop\OSP2C.exe
Lloji i pool te mesazheve eshte System.Runtime.Remoting.Channels.BinaryClientFormat
terSink.
Klienti po therret remote object.
Remote objecti eshte thirrur 2 here.
Deshironi t'e perseritni? (1-po, 0=jo):
Klienti po therret remote object.
Remote objecti eshte thirrur 4 here.
Deshironi t'e perseritni? (1-po, 0=jo):
Klienti po therret remote object.
Remote objecti eshte thirrur 7 here.
Deshironi t'e perseritni? (1-po, 0=jo):
Klienti po therret remote object.
Remote objecti eshte thirrur 10 here.
Deshironi t'e perseritni? (1-po, 0=jo):
Klienti po therret remote object.
Remote objecti eshte thirrur 13 here.
Deshironi t'e perseritni? (1-po, 0=jo):
Klienti po therret remote object.
Remote objecti eshte thirrur 17 here.
Deshironi t'e perseritni? (1-po, 0=jo):

C:\Users\agoni\Desktop\OSP2S.exe
Lloji i pool te mesazheve eshte System.Runtime.Remoting.Channels.BinaryClientFormat
terSink.
Klienti po therret remote object.
Remote objecti eshte thirrur 1 here.
Deshironi t'e perseritni? (1-po, 0=jo):
Klienti po therret remote object.
Remote objecti eshte thirrur 6 here.
Deshironi t'e perseritni? (1-po, 0=jo):
Klienti po therret remote object.
Remote objecti eshte thirrur 9 here.
Deshironi t'e perseritni? (1-po, 0=jo):
Klienti po therret remote object.
Remote objecti eshte thirrur 12 here.
Deshironi t'e perseritni? (1-po, 0=jo):
Klienti po therret remote object.
Remote objecti eshte thirrur 15 here.
Deshironi t'e perseritni? (1-po, 0=jo):
Klienti po therret remote object.
Remote objecti eshte thirrur 18 here.
Deshironi t'e perseritni? (1-po, 0=jo):

```

Figura 2. Disa klient të qasur në një server të vetëm

Po t'i shikonim vetëm klientët, do mund të mendonim se këtu nuk kemi më konsistenc, mirëpo kur t'e shohim nga ana e serverit, qdo thirrje është bërë me radhë, dhe ka pasur inkrementim vetëm një herë qdo rast kur një klient ka pasur qasje në metodën e objektit.

Kjo zgjidhje nuk është e limituar në aplikim vetëm në një kompjuter. është një zgjidhje që mund të e përdorim në një rrjet me disa modifikime të vogla – klienti kur të kërkoj qasje në metod, do duhej të e kërkonte pastaj në IP të serverit (qoftë private për rrjet lokale, apo reale për rrjet të gjerë), dhe jo në localhost, siq e kërkon tani. Pra, me një modifikim të vogël, do mund të e aplikonim këtë zgjidhje në mjedis komplet të ndryshëm dhe vërtetë të e bënim një server dhe klient.

Në vazhdim do e shtjellojmë edhe një shembull më të thjeshtë, paksa më të limituar dhe të bazuar në konceptin prodhues-konsumues, dhe kësaj herë si komunikim mes threads të një procesi të vetëm. Shfaqjen e një zgjidhje të dytë e shohim të arsyeshme që të tregohet se si message passing mund të përdoret edhe për komunikim interthread, jo vetëm interprocess.

Zgjidhja 2 – Komunikim ndërmjet threads me message passing

```
package test;
import java.util.Vector;
class Server extends Thread { //Serveri i shkruar nga: Agon Hoxha
    static final int MAXQUEUE = 5; //5 mesazhe maksimum ne queue
    private Vector messages = new Vector();
    public void run() {
        try {
            while (true) {
                putMessage(); //thirret funksioni putMessage
                sleep(1000); //qdo 1 sekond
            }
        } catch (InterruptedException e) {}
    }
    private synchronized void putMessage() throws InterruptedException {
        while (messages.size() == MAXQUEUE) //derisa te mos ka mesazhe ne queue
            wait(); //thread wait state
        messages.addElement(new java.util.Date().toString()); //dergo datetime
        notify(); //ringjall threadin nga gjendja e pauzuar
    }
    // Thirre kit funksion prej klientit per me marr mesazhin
    public synchronized String getMessage() throws InterruptedException {
        notify();
        while (messages.size() == 0) //gjersa nuk i ka ardhur asnje mesazh
            wait();
        String message = (String) messages.firstElement(); //Merre mesazhin
        messages.removeElement(message); //largohet mesazhi nga queue
        return message; //kthen mesazhin
    }
}
class Client extends Thread { //Klienti i shkruar nga: Amire Gerguri
    Server server; //krijim i objektit server, te tipit Server ^
    Client(Server s) { //inicimi i klases Klienti me server ne konstruktor
        server = s;
    }
    public void run() {
        try {
            while (true) {
                String message = server.getMessage(); //e merr mesazhin
                System.out.println("Mesazhi i marrur: " + message);
                // ^ printon mesazhin e marrur
                System.out.println("Tani duke pritur.");
                sleep(2000);
            }
        } catch (InterruptedException e) { System.out.println("Error ka ndodhur."); }
    }
    public static void main(String args[]) { //pika e fillimit te ekzekutimit
        Server server = new Server(); //krijimi i objektit server
        server.start(); //krijohet thread te serverit
        new Client(server).start(); //krijohet klient me server ne konstruktor
    }
}
```


Zgjidhja 2 – Shpjegim

Qka vërejmë më së pari tek ky aplikacion është se është vetëm një executable. Nuk kemi dy aplikacione të ndara si në rastin paraprak, dhe kompleksiteti i saj është tejet më i vogël.

Një tjetër gjë që vërejmë, kur t’ë ekzekutojmë është se nuk varet në ndonjë mënyrë nga input të përdoruesit. Gjersa zgjidhja paraprake i duhej përdoruesit që të shtyp ndonjë tast apo të ndërmerr ndonjë veprim, në këtë rast, nuk ka nevoj fare.

Në rastin e shfaqur, dërgohet mesazhi prej server threadit tek klient threadi dhe pauzohet ekzekutimi, vazhdon ekzekutimi me threadin e klientit – i cili vetëm ka për detyrë të kontrolloj se a ka ndonjë mesazh, dhe t’ë printoj atë. Sapo t’ë printoj, threadi i klientit vendoset në gjendjen e paузuar dhe riaktivizohet threadi i serverit, i cili prap vendos mesazh në queue, dhe kështu vazhdon.

Si mesazh në këtë rast është zgjedhur data dhe koha – kjo për arsye që të shohim ndonjë ndryshim në mesazh, dhe të sigurohemi se vërtetë po ka komunikim dhe nuk jemi thjeshtë duke e printuar manualisht.

```
Mesazhi i marrur: Sat Jan 05 22:10:56 CET 2019
Tani duke pritur per mesazhin e radhes.
Mesazhi i marrur: Sat Jan 05 22:10:57 CET 2019
Tani duke pritur per mesazhin e radhes.
Mesazhi i marrur: Sat Jan 05 22:10:58 CET 2019
Tani duke pritur per mesazhin e radhes.
Mesazhi i marrur: Sat Jan 05 22:10:59 CET 2019
Tani duke pritur per mesazhin e radhes.
Mesazhi i marrur: Sat Jan 05 22:11:00 CET 2019
Tani duke pritur per mesazhin e radhes.
Mesazhi i marrur: Sat Jan 05 22:11:01 CET 2019
Tani duke pritur per mesazhin e radhes.
Mesazhi i marrur: Sat Jan 05 22:11:02 CET 2019
Tani duke pritur per mesazhin e radhes.
Mesazhi i marrur: Sat Jan 05 22:11:03 CET 2019
Tani duke pritur per mesazhin e radhes.
Mesazhi i marrur: Sat Jan 05 22:11:04 CET 2019
Tani duke pritur per mesazhin e radhes.
```

Figura 3. Rezultati i ekzekutimit

Në këtë rast, pika kyqe e programit është përdorimi i klasës Vector. Me anë të kësaj klase, krijojmë një objekt në të cilin mund të vendosim qkado. Në rastin tonë, pasi që e kemi formatin të pacaktuar, tipi i Vector messages do jete raw – që teoretikisht do na lejonte të dërgonim qkado, duke përfshire edhe stringje të thjeshta. Sidoqoftë, tipi raw do duhej të anashkalohej.

Threadi i serverit kurdo që thirret apo riaktivizohet në gjendjen e ekzekutimit, merr kohën dhe datën momentale, dhe e vendos atë në objektin messages të tipit Vector, nëse ka vend në të. Për vektor aplikohet kufizimi i numrit të antarëve, që tek në është 5. Pra 5 mesazhe mund të vendosen në queue, para se të ndalohej shtuarja e mëtejshme.

Threadi i klientit kurdo që fillon ekzekutimin, thërret funksionin e getMessage të serverit, i cili gjeneron një mesazh dhe e vendos atë në objektin messages, dhe pauzon threadin e vet, duke kthyer kontroll në threadin e klientit, i cili pastaj e shfaq mesazhin e parë që gjendet në objektin messages. E vendos threadin e vet në gjendje të paузuar për një kohë të caktuar, para se të rifilloj procedurën prap.

Ky program është i bazuar në konceptin e thjeshtë prodhues-konsumator – një program, apo pjesë e programit krijon disa të dhëna, tjetri program apo pjesë e programit, manipulon me ato të dhëna. Ky proces që përsëritet deri në pafundësi, apo derisa të plotësohet ndonjë kusht, i caktuar nga programerët. Në këtë rast nuk ka ndonjë kusht që do shkaktonte ndaljen e ekzekutimit të programit, andaj ai nuk do ndalej fare.

Referencat

Dokumentacioni publik për klasat e përdorura në C#:

- <https://docs.microsoft.com/en-us/dotnet/api/system.security.permissions.securitypermission>
- <https://docs.microsoft.com/en-us/dotnet/api/system.runtime.remoting> dhe nënklasat:
 - o <https://docs.microsoft.com/en-us/dotnet/api/system.runtime.remoting.channels.ipc.ipcchannel>
 - o <https://docs.microsoft.com/en-us/dotnet/api/system.runtime.remoting.channels.channelservices>
 - o <https://docs.microsoft.com/en-us/dotnet/api/system.runtime.remoting.channels.channeldatastore>
 - o <https://docs.microsoft.com/en-us/dotnet/api/system.runtime.remoting.messaging.imessagesink>

Dokumentacioni publik për klasat e përdorura në Java:

- <https://docs.oracle.com/javase/8/docs/api/java/lang/Thread.html>
- <https://docs.oracle.com/javase/8/docs/api/java/util/Vector.html>
- <https://docs.oracle.com/javase/tutorial/essential/concurrency/syncmeth.html>

Informata tjera në lidhje me domenin e detyrës:

- https://en.wikipedia.org/wiki/Message_passing
- <https://docstore.mik.ua/orelly/java-ent/dist/index.htm>

Materiali i përpunuar përgjatë semestrit:

- Ligjerata 6 & 7 - Isak Shabani
- Ushtrimet 5, 7 & 8 - Artan Mazrekaj

Materiali bazë i lëndës:

- Operating System Concepts, 10th Edition, 2018 – Abraham Silberschatz, Peter Baer Galvin & Greg Gagne