**Local Ağdaki Bilgisayarlar Arası Haberleşme Uygulaması**

**ETHEM ATEŞ**

**HACETTEPE ÜNİVERSİTESİ BİLGİSAYAR MÜHENDİSLİĞİ**

**İÇERİKLER**

KAPAK 1

ÖZET 3

KÜTÜPHANELER 4

KODUN GÖRSEL ŞEMASI 7

FONKSİYONLARIN AÇIKLAMASI 8

MAİN FONKSİYON VE FONKSİYONLARIN KULLANIMI 13

**ÖZET**

13.09.2024

Local Ağdaki Bilgisayarlar Arası Haberleşme Uygulaması,

Bu kod, canlı ses iletişimi gerçekleştirmek amacıyla mikrofonla alınan ses verilerini internet üzerinden karşı tarafa iletmek ve gelen sesleri çalmak için tasarlanmıştır. Mikrofon ile alınan sesler önce işlenip belirli bir formatta kaydedilir, ardından verimlilik sağlamak için ses verileri Opus Codec kullanılarak sıkıştırılır. Bu sıkıştırılmış veriler UDP protokolü kullanılarak karşı tarafa gönderilir. Aynı şekilde karşı taraftan gelen sıkıştırılmış ses verileri alınır, çözülür ve hoparlörden oynatılır.

Kodun amacı, düşük gecikmeyle ve kaliteli bir şekilde iki taraf arasında ses iletişimi sağlamaktır. Verilerin düzgün işlenmesi ve ağ üzerinden kayıpsız bir şekilde iletilmesi için güvenli bir sıra yapısı (SafeQueue) ve çoklu iş parçacığı kullanılır. Bu yapı, mikrofon girişinden gelen verilerin kaydedilmesini ve alınan ses verilerinin aynı anda çözülüp oynatılmasını mümkün kılar. Ek olarak, Opus Codec ile sıkıştırma işlemi sayesinde ses verileri minimum bant genişliği kullanılarak iletilir ve ses kalitesi korunur.

Sesin düzgün iletilmesi ve alınmasındaki gecikme ve veri kayıplarını azaltmak için UDP protokolü tercih edilmiş ve Opus Codec ile ses kalitesi optimize edilmiştir. Bu yapılar birlikte, özellikle düşük gecikmeli ses iletişimi için etkili bir çözüm sunar.

**KÜTÜPHANELER**

|  |  |
| --- | --- |
| **Kullanılan External Kütüphaneler** | |
| **Kütüphane Adı** | **Kullanım Amacı** |
| Opus | Kaydedilen verileri sıkıştırmak ve çözmek için kullanılır |
| Portaudio | Sesli verileri almak ve kaydetme süreçleri için kullanılır |
| Asio | Verilerin internet üzerinden iletimini sağlamak için kullanılır |

**Portaudio:** Portaudio kullanmamızın en temel sebebi sesli iletişim uygulamalarının temeli olan sesi kaydetme ve tüm bu işlem sırasında süregelen alt işlemleri yönetmek için kullanılır. Bilgisayardaki tüm ses kartlarına erişimi ve onların özelliklerini listelemesi read ve playback fonksiyonu yazma imkanı sağlaması sebebiyle de öne çıkar. Kod dosyasında da görüleceği üzere “err” adında bir PaError nesnesi oluşturulur ve tüm bu süreç bu nesnenin üzerinden gerçekleşir. Koda eklediğim “checkError” fonksiyonu her süreçte hata gerçekleşip gerçekleşmediğini kontrol eder eğer hata gerçekleşmişse bunun mesajını döndürür. Temel olarak bir ses kaydetme işlemi 5 aşamadan oluşur. Yani “err” nesnesi üzerinde 5 farklı işlem gerçekleşir ve bu 5 noktada aynı zamanda hata oluşumu kontrol edilir. Bu işlemler sırasıyla: “Pa\_Initialize()” , “Pa\_OpenStream()” , “Pa\_StartStream()”, “Pa\_StopStream()” , “Pa\_CloseStream()”. Bu 5 işlemi ya kod üzerinde yorum satırı olarak ya da raporun devamında açıklayacağım. Tüm bu işlemlerin devamında da “Pa\_Terminate()” işlemi ile sonlandırılır. Portaudio tüm bu işlemlerden önce Stream oluşturmada input ve output parametresi oluşturmada ve sesi içinde bulunduran bir struct yapısı olan “PaTestData” yapısını oluşturmada kullanılır. Hem işlemlerdeki kolaylığı hem de dokümantasyonunun genişliği ve kalitesi sayesinde diğer kütüphanelerden biraz daha öne çıkıyor ve onu kullanmaya değer yapıyor.

İndirme Linki: [**https://files.portaudio.com/download.html**](https://files.portaudio.com/download.html)

Versiyon: 19.7

metin, ekran görüntüsü, yazı tipi, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Opus:** Opus Codec kütüphanesinin kullanılmasının sebebi kaydedilen sesin sıkıştırma(encode) ve çözme(decode) işlemlerini gerçekleştirmektir. Sıkıştırma ve çözme işlemleri ağ üzerinden gönderilen verilerin ağı yormaması için gereklidir. Bu işlevi yapabilecek çok çeşitli kütüphaneler c++ için yok. Elimizde olan ve basit kullanımıyla öne çıkan Opus Codec, bu işlemlerin amiral gemilerinden biri, benim kullanma sebebim ise sadece sıkıştırma ve çözme işlemlerinin yanı sıra sesin özelliklerini ayarlama konusunda bize avantajlar sağlamasıydı. Bu özellikleri kodları encoder temelinde kullandım ve kodu incelediğinizde “ opus\_encoder\_ctl()” fonksiyonları olarak görebilirsiniz. Özellikle düşük bit hızlarında yüksek kaliteli ses elde etmek için optimize edilmiştir. VoIP (Voice over IP) uygulamaları, video konferanslar ve ses akışı gibi birçok alanda yaygın olarak kullanılmaktadır. Skype, Discord, Zoom, Teams gibi uygulamalarda da kullanılır. Farklı bit hızları ve kalite ayarlarına izin verir. Bu sayede farklı kullanım senaryolarına uyum sağlayabilir. Kullanımında encoder decoder oluştururken bizim kontrol değerimiz olan OPUS\_OK ile encoder veya decoder hata kontrolü yapıyorum. Encoder ve decoder oluşturmak ve daha sonra oluşturduğum nesnelerle “sendAudio” ve “receiveAudio” dosyalarının içinde “opus\_decode\_float()” ve “opus\_encode\_float()” fonksiyonlarını kullanarak elimde olan veriyi “unsigned char” nesnesine sürekli olarak sıkıştırıp ya da oradaki veriyi sürekli olarak çözüp veriyi iletmeye veya dinlemeye hazır hale getiriyorum. Bu işlemi hata kontrolü için bir “integer” değerine bağlıdır o yüzden eğer 0’dan küçük bir değer return lediğinde de decode ya da encode işlemindeki hatayı görebiliyorum. Bunla ilgili detayları kod üzerindeki yorum satırlarında bulabilirsiniz. Ve tabii ki “opus\_encoder\_ctl()” fonksiyonlarını kullanarak sesi istediğim şekilde yapılandırıyorum. Ben versiyon olarak 1.5.1 kullanıyorum vcpkg’ de hazır olarak 1.5.1 var ama internet sitesinde en yeni sürüm 1.5.2 build issue fix dışında herhangi bir değişiklik yok yeni sürümde. Benim kodum herhangi bir build issue vermiyor o yüzden kullanımında bir değişiklik yaratmayacaktır.

İndirme Linki: [**https://opus-codec.org/downloads/**](https://opus-codec.org/downloads/)

metin, ekran görüntüsü, yazı tipi, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Asio:** Asio kütüphanesinin kullanımdaki görevi alınan verileri internet üzerinden alıp iletmektir. Diğer kütüphanelerden farkı modern ağ programlamasının karmaşıklıklarını büyük ölçüde basitleştirerek geliştiricilere güçlü bir altyapı sunar. Kullanımı bu alanda çok çalışmamış kişiler için görece kolaydır. Kurulan soketlerle birden fazla giriş çıkış işlemini aynı anda yönetebilme imkanı tanır. Birçok protokolü destekler ve kullanıcılar için geniş bir topluluk desteği sunar. Öncelikle her iki bilgisayar için “sender” ve “receiver” socketlerini kurdum. Ve tüm işlemler bu iki soket arasında gerçekleştirdim.

İndirme Linki: [**https://think-async.com/Asio/**](https://think-async.com/Asio/)

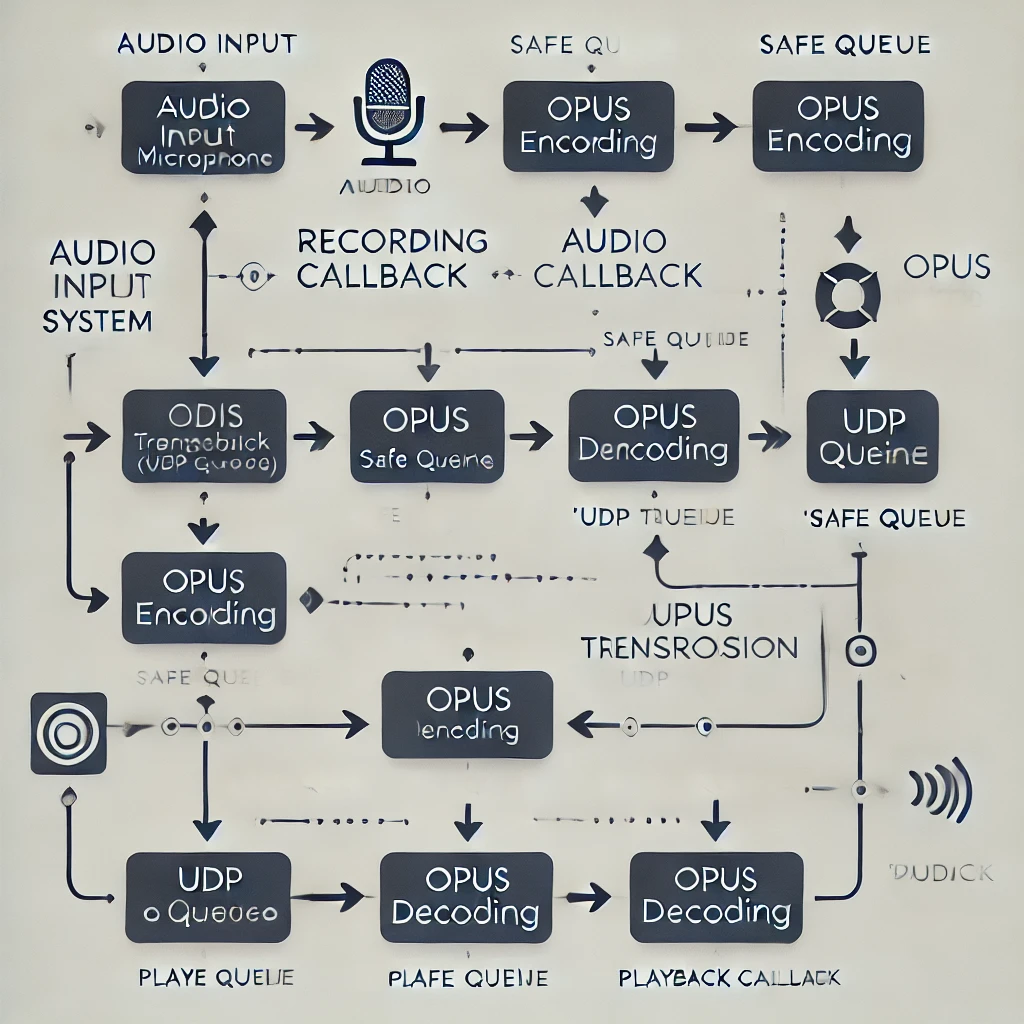
**metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**metin, yazı tipi, yazılım, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**KODUN GÖRSEL ŞEMASI:**

****

**FONKSİYONLARIN AÇIKLAMASI:**

**#define bölümleri**

**metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**SAMPLE\_RATE**: ses örnekleme hızını saniyede 48.000 örnekleme olarak ayarlar.

**FRAMES\_PER\_BUFFER**: bir Bufferdaki sample(örnek) sayısını 1920 olarak ayarlar. Tamponlar sesin geçici olarak depolandığı yerdir. Daha büyük tampon daha az sıklıkta veri işlenmesini gerektirdiği için bozulma azalır ancak daha fazla bellek tüketir.

**NUMBEROFCHANNELS:** Kanal sayısını belirtir. Stereo ses için kanal sayısı 2’dir. Mono ses için 1 kanal sayısı kullanılır.

**PA\_SAMPLE\_TYPE:** Ses örneklerinin veri türünü 32-bit kayan nokta olarak tanımlar. Çoğu örnek veri türünün kullandığı değer paFloat32’dir.

**MAX\_PACKET\_SIZE:** Bir ağ paketindeki maksimum veri boyutunu temsil eder. Bir ağ paketi, ağ üzerinden gönderilen veri birimidir. Daha büyük bir paket boyutu, daha az sıklıkta paket gönderilmesini gerektirir ancak daha fazla ağ trafiği oluşturur.

**PORT:** Portu 12345 olarak tanımladım. Bir port, bir bilgisayarın ağ üzerindeki bir uygulamanın dinlediği bir adrestir. IP adresi evse port da o eve giriş kapılarıdır.

**IP\_ADDRESS:** Burada ayarlanan IP adresi her bilgisayar için kendi ip adresini temsil eder. Kodumda sesi dinlemek için kendi ip’m den kendi ip’me yolladığım için tek ip adresi yeterli oluyor. Ama iki bilgisayar arasında bu iletişim olacağı için kodun düzenlenmesinde buraya diğer bilgisayarın ip adresinin de eklenmesi gerek.

**BITRATE**: Bu satır, ses akışının bit hızını saniyede 128.000 bit olarak tanımlar. Bit hızı, ses verilerinin saniyede kaç bit ile aktarıldığını belirtir. Daha yüksek bir bit hızı, daha yüksek ses kalitesi sağlar.

**#safeQueue ve paTestData**

metin, ekran görüntüsü, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

Kodumuzda bir safeQueue yapısı kullanılmak zorundadır. Çünkü bir thread ses verisini kaydeden, diğer thread ise bu veriyi çalan olduğu için bu yapı klasik bir "producer-consumer" problemini çözer. Producer, kaydedilen ses verisini sıraya alır, consumer ise bu sıradan veriyi çekip oynatır. Aynı zamanda çoklu thread'lerin sıraya erişimi esnasında veri bozulmasını önlemek için bir kullanılır. Condition variable, sıranın boş ya da dolu olduğu durumlara göre thread'lerin bekletilmesini sağlar.

paTestData yapısı ses kaydı ve oynatma işlemleri sırasında veriyi kuyrukta tutmak için tanımlanan bir yapıdır. safeQueue, kaydedilen ses verilerini tutar ve bu veriler daha sonra oynatma işlemi sırasında kullanılır.

**checkError :**

**ekran görüntüsü, metin içeren bir resim

Açıklama otomatik olarak oluşturuldu**

Portaudionun işleyiş verisi olan PaError nesnesinde hata oluşup oluşmadığını kontrol eder. Eğer hata tespit ederse hatanın içeriğini konsola yazdırır.

**recordCallbackFunction :**

metin, ekran görüntüsü, yazılım, multimedya yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu

Burada görülen fonksiyonun temel amacı sesi kaydetmektir buffer’a sesi alan fonksiyon budur. Buffer vektörü, alınan ses verilerini geçici olarak saklamak için kullanılır. if (inputBuffer == NULL) bloğu, inputBuffer'ın boş olup olmadığını kontrol eder.Boşsa, buffer vektörü sıfır değerleriyle doldurulur. İç döngü, inputBuffer'daki her bir ses örneğini alır ve buffer vektörüne ekler. Oradaki 0.8 değeri sesin 0.8 katı anlamna gelir sesi arttırmak için burayı arttırmak mantıklı olacaktır. İşlem sırasında, ses seviyesini ayarlamak için bir çarpma işlemi yapılır ve sonuç -1 ile 1 arasında olacak şekilde sınırlandırılır. (birçok ses kaynağında ses verileri 1 ile 1 arasında standarlaştırılmıştır.)

Data->queue.enqueue(buffer) satırı, işlenmiş verileri data->queue isimli kuyruğa ekler.

**playCallbackFunction :**

**metin, ekran görüntüsü, yazılım, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu**

Bu fonksiyonun görevinin en temel açıklaması kayıtlı olan sesi çalmasıdır. bir ses çalma işlemi sırasında çağrılan bir geri çağırma (callback) fonksiyonunu tanımlar. Bu fonksiyon sesi otomatik olarak karttan çalar. data->queue isimli bir kuyruktan ses verilerini alır. Alınan ses verilerini, outputBuffer parametresi aracılığıyla ses kartına gönderir. Bufferdaki veriler “writer” ile outputbufferına veriler yazılır. recordCallbackFunction ve playCallbackfunction daki kullanılmayan parametreler “Pa\_OpenStream” ile doldurulur dolayısıyla fonksiyonumuzda parametre olarak kalmak zorundadır.

**sendAudio:**

**metin, ekran görüntüsü, yazılım, işletim sistemi içeren bir resim

Açıklama otomatik olarak oluşturuldu**

Bu fonksiyon, gerçek zamanlı ses iletimi için kullanılır. Queue’dan birer ses frame i alarak Opus ile sıkıştırıyor. Sıkıştırılmış veriler UDP soketi aracılığıyla belirli bir alıcıya gönderiliyor. Sürekli olarak veri alıp sıkıştırıp gönderdiği için bu işlem döngü şeklinde devam ediyor. “compressedSize” integer’ı hata vermesi durumunda yani 0’dan düşük olması durumunda opus sıkıştırma hatası verir. Gönderimdeki hatalar asio error’ıyla kontrol edilir.

**receiveAudio:**

**metin, ekran görüntüsü, yazılım, multimedya yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu**

Bu fonksiyon temel olarak sesi düzenli olarak almak için kurulmuştur. Öncelikle, “receive\_buffer” isimli bir vektör, maksimum paket boyutunu alacak şekilde ayrılır. Bu vektör, gelen sıkıştırılmış ses verilerini tutacaktır. “receive\_socket.receive\_from” fonksiyonu, UDP soketi üzerinden veri alma işlemini başlatır. Bu fonksiyon, karşı tarafın IP adresi ve port numarasını da “remote\_endpoint” değişkenine atar. Eğer veri alma işleminde bir hata oluşursa ve hata mesaj boyutu ile ilgili değilse hata mesajı yazdırılır ve döngüye devam edilir. Gelen sıkıştırılmış veriler “receive\_buffer” vektöründe tutulduktan sonra, Opus kullanılarak bu veriler, orijinal ses örneklerine dönüştürülür. Dönüştürülen ses örnekleri, “decodedFrame” vektöründe saklanır. Kod çözme işleminde hata oluşursa, hata mesajı yazdırılır ve döngüye devam edilir. Son olarak, dönüştürülmüş ve işlenmiş ses sampleları, receiveQueue isimli queue’ya eklenir. Bu kuyruk, daha sonra çeşitli ses işlemlerinde kullanmak için saklanır.

**MAİN VE FONKSİYONLARIN KULLANIMI:**

**OLUŞTURMA AŞAMALARI:**

**metin, ekran görüntüsü, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu**

PaError err = paNoError: Bu satır, PortAudio kütüphanesinden bir hata değişkeni tanımlar.

Encoder ve decoder tanımlanır bunların oluşumundaki hata varlığı OPUS\_OK değişkeniyle kontrol edilir. Bir asio::io\_context nesnesi oluşturulur. Bu nesne, Asio'nun asenkron işlemlerini yönetmek için kullanılır. asio::ip::udp::socket soketi oluşturulur ve Bu soket, ses verilerini göndermek için kullanılır. asio::ip::udp::socket soketi oluşturulur ve Bu soket, ses verilerini almak için kullanılır.

**ENCODER CTL AYARLARI:**

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**opus\_encoder\_ctl(encoder, OPUS\_SET\_BITRATE(BITRATE));**: Bu satır encoder ın bitrate’ini ayarlar biz bu veriyi 128000 olarak aldık. Daha yüksek bitrate daha iyi ses kalitesi demektir ama daha fazla bant genişliği gerektirir.

**opus\_encoder\_ctl(encoder, OPUS\_SET\_BANDWIDTH(OPUS\_BANDWIDTH\_FULLBAND));**: Bu satır bant genişliğini tüm bant genişliği olarak alır. (tam bant kullanılır)

**opus\_encoder\_ctl(encoder, OPUS\_SET\_COMPLEXITY(10));**: Encoder’ın karmaşıklık seviyesini 10’a ayarlar bu değer yüksek bir değerdir. Daha yüksek karmaşıklık daha iyi ses kalitesi demektir ama daha fazla işlem gücü gerektirir.

**opus\_encoder\_ctl(encoder, OPUS\_SET\_SIGNAL(OPUS\_SIGNAL\_VOICE));**: Bu kod encoder’ın konuşma sinyallerine optimize edilmiş ayarlar kullanmasını sağlar. Böylece kısmi bir gürültü azaltma uygulanmış olur.

**opus\_encoder\_ctl(encoder, OPUS\_SET\_INBAND\_FEC(1));**: Bu satır paket kaybı durumunda ses kalitesini iyileştirmesini sağlar.

**opus\_encoder\_ctl(encoder, OPUS\_SET\_PACKET\_LOSS\_PERC(10));**: Bu satır encoder’a hatanın yani paket kaybının yüzde 10 olduğunu söyler ve encoder’a önlem almasını söyler

**opus\_encoder\_ctl(encoder, OPUS\_SET\_DTX(1));**: Bu satır DTX kullanımını yani sessiz zamanda daha az veri gönderimini teşvik eder.

**İNPUT OUTPUT VE STREAM PARAMETRELERİNİN OLUŞTURULMASI:**

metin, ekran görüntüsü, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

**err = Pa\_Initialize();:** PortAudio kütüphanesini başlatır. Başlatma işleminde hata olursa err değişkenine hata kaydedilir.

**checkError(err);:** Bu fonksiyon, err değişkenindeki hata kodunu kontrol eder. Eğer hata varsa, hata mesajı ekrana yazdırılır ve program sonlandırılır.

**PaStream\* inputStream**;: Bir PaStream nesnesi için bir işaretçi tanımlar. Bu nesne, ses giriş akışını temsil edecektir.

**paTestData inputData;:** Giriş akışı için özel veri tutmak için bir yapı tanımlar.

**PaStreamParameters inputParameters**;: Giriş akışı parametrelerini tanımlamak için bir yapı tanımlar.

**device**: Giriş cihazını belirtir. Pa\_GetDefaultInputDevice() fonksiyonu, varsayılan giriş cihazını alır.

**channelCoun**t: Kanal sayısını belirtir. Stereo için 2.

**sampleFormat**: Sample formatı örneğin, paFloat32

**suggestedLatency**: Önerilen gecikmeyi belirtir. Pa\_GetDeviceInfo fonksiyonu, cihazın önerilen düşük gecikme değerini alır.

**hostApiSpecificStreamInfo**: İnternetteki örneklerinde null olarak

ayarlandığı için null olarak ayarladım.

**PaStream\* outputStream**;: Bir PaStream nesnesi tanımlar. Bu nesne, ses çıkış akışını temsil edecektir.

**paTestData outputData;:** Çıkış akışı için özel veri tutmak için bir yapı tanımlar.

**PaStreamParameters outputParameters**;: Çıkış akışı parametrelerini tanımlamak için bir yapı tanımlar. İçindeki parametreler **inputParameters** ile aynıdır.

**AKIŞIN BAŞLATILMASI, TAKİBİ, DURDURULMASI VE KAPATILMASI:**

metin, ekran görüntüsü, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Pa\_OpenStream**: Giriş, çıkış streamlerini açar. Parametreler olarak, stream nesnesi, giriş/çıkış parametreleri, örnekleme hızı, tampon boyutu, sabit olarak(paClipOff), geri çağırma fonksiyonu ve input yapısını alır.

**Pa\_StartStream**: Giriş ve çıkış streamlerini başlatır

**thread sendThread**: Bir ithread oluşturur ve sendAudio fonksiyonunu çalıştırır. Bu fonksiyon, ses verilerini göndermekten sorumludur.

**thread receiveThread**: Bir thread oluşturur ve receiveAudio fonksiyonunu çalıştırır. Bu fonksiyon, ses verilerini almaktan sorumludur.

**sendThread.join();:** sendThread’in işini tamamlamasını bekler**.**

**receiveThread.join();** receiveThread ‘in işini tamamlamasını bekler.

**Pa\_StopStream**: Giriş, çıkış streamlerini durdurur.

**Pa\_CloseStream**: Giriş ve çıkış streamlerini kapatır.

**opus\_encoder\_destroy(encoder);:** Opus encoderı yok eder.

**opus\_decoder\_destroy(decoder);:** Opus decoderını yok eder.

**Pa\_Terminate();:** PortAudio kütüphanesini sonlandırır.