تمرین کامپیوتری شماره 1

عنوان: Distributed voice call using WebRTC

درس: شبکه‌های کامپیوتری

استاد راهنما: دکتر ناصر یزدانی

رشته: مهندسی کامپیوتر

دستیاران آموزشی: سهراب مرادی[۱]، هستی کریمی[۲]، محمدرضا ولی[۳]

نیمسال اول سال تحصیلی 04-1403

[۱] m.moradi@ut.ac.ir
[۲] hasti.karimi@ut.ac.ir
[۳] mvali@ut.ac.ir

عنوان پروژه

تماس صوتی زنده توزیع شده (Socket Programming – distributed live voice call)

هدف

در این تمرین قصد داریم با استفاده از ابزارهای رایج به پیاده سازی یک سیستم توزیع شده ی برقراری تماس زنده بپردازیم. یک سیستم ساده متشکل از 2 دستگاه که بدون اتکا به سرور مرکزی با یکدیگر تبادل داده لحظه ای به شکل صدا انجام می دهند.

0- ابزار

این تمرین تماما با استفاده از زبان برنامه نویسی C++ پیاده سازی خواهد شد لذا از کتابخانه ها و فریمورک های مطرح و محبوب این زبان استفاده خواهیم کرد. ابزار مورد نیاز به شرح زیر است:

- Qt framework
- WebRTC (with libdatachannel)
- کدک صوتی Opus
- Coturn (امتیازی)

1- مقدمه

در ابتدا لازم است تا ابزار مورد نیاز را بر روی سیستم خود نصب کنید. تمامی موارد معرفی شده cross-platform هستند و بدون در نظرگرفتن سیستم عامل می توانید از آن ها استفاده کنید.

1-1- نصب فریمورک Qt

برای نصب این ابزار با مراجعه به مستندات رسمی Qt گام های مربوط به ایجاد حساب کاربری و دانلود نصب کننده ی آنلاین را انجام دهید (دقت نمایید که حتما نسخه ی open source را از این لینک (لینک دانلود نصب کننده ی کیوت) تهیه نمایید در غیر این صورت مجبور به دانلود و نصب مجدد خواهید شد) و سپس مطابق با تصویر زیر پکیج های مورد نیاز را انتخاب و نصب کنید. دقت شود در تصاویر زیر فقط پکیج هایی که تیک سبز دارند انتخاب شوند. دقت کنید که برای دانلود پایدار و بدون مشکل حتما از ابزار های گذر از تحریم مانند شکن استفاده نمایید. هنگام نصب با نسخه های متفاوتی از Qt مواجه خواهید شد، پیشنهاد ما برای نصب روی ویندوز ورژن 6.5.3 و برای لینوکس نسخه (در صورت عدم وجود نسخه ی 6.5.3 نسخه ی 6.7.3 است). پس از انجام تمرین اول، این فریمورک را از سیستم خود حذف نکنید زیرا برای تمرین 3 نیز به آن احتیاج خواهید داشت.

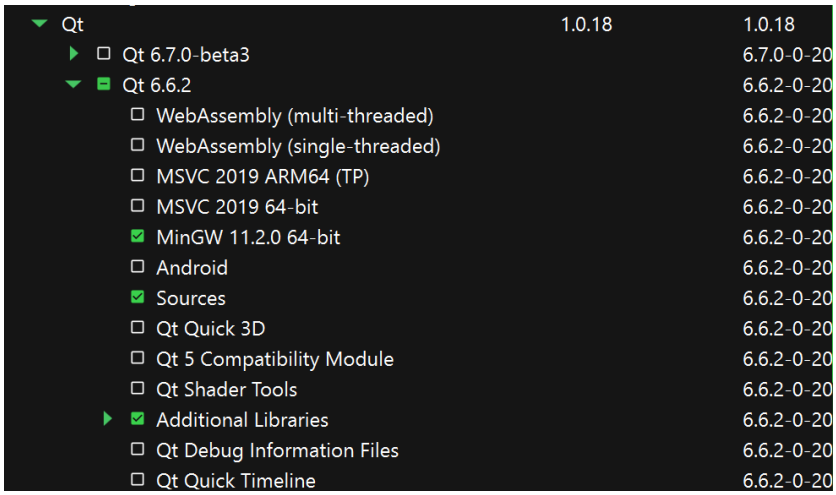

Figure 1. پکیج های لازم بخش اول

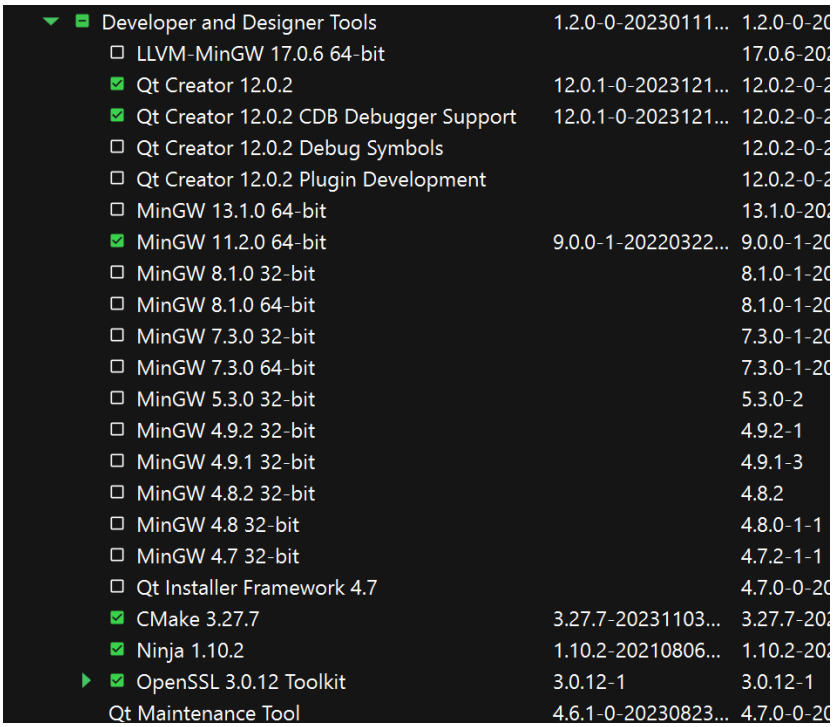

Figure 2. پکیج های مورد نیاز بخش دوم

برای ویندوز حتما جدید ترین نسخه ی mingw را تیک بزنید، در لینوکس نیز ابزار gcc , g++ را نصب کنید:

sudo apt update

sudo apt install build-essential

g++ --version

درصورت تمایل میتوانید بر روی ویندوز از MSVC استفاده نمایید، برای این منظور باید این کامپایلر را بوسیله ی نصب کننده ی visual studio دانلود و نصب کرده و همچنین در حین نصب کیوت، تیک MSVC 2019 را نیز بزنید. وبسایت رسمی مایکروسافت برای ابزار های visual studio

1-2- نصب کتابخانه ی libdatachannel برای استفاده از پروتکل WebRTC

این پروتکل به منظور ارتباط peer to peer توسط شرکت گوگل توسعه داده شده است که متاسفانه راه ساده ای برای استفاده از این پروتکل به وسیله ابزار های رسمی بر روی C++ وجود ندارد (راه اصلی کامپایل سورس کد این کتابخانه است که به دلیل تحریم و تعداد زیاد پیش نیاز ها از آن اجتناب می کنیم). به همین جهت برای استفاده سریع و آسان از این پروتکل در CPP از کتابخانه ی libdatachannel استفاده می کنیم. برای این منظور شما باید کد منبع این کتابخانه از گیت هاب آن کلون نمایید و نصبت به build کردن آن اقدام کنید (مراحل build روی ویندوز):

git clone https://github.com/paullouisageneau/libdatachannel.git

cd libdatachannel

git submodule update --init --recursive --depth 1

cmake -B Windows/Mingw64 -DUSE_GNUTLS=0 -DUSE_NICE=0 -DCMAKE_BUILD_TYPE=Release -DOPENSSL_ROOT_DIR=C:\Qt\Tools\OpenSSLv3\Win_x64 -G "MinGW Makefiles"

cmake --build Windows/Mingw64

راهنمای build بر روی دیگر سیستم عامل ها

پس از build کردن این کتابخانه نوبت به لینک کردن آن در پروژه میرسد: (پیشنهاد می شود در حین ساخت یک پروژه ی جدید کیوت، از QMake به عنوان build system استفاده نمایید)

```
### in file PROJECT_NAME.pro
INCLUDEPATH += $$PATH_TO_LIBDATACHANNEL /include
LIBS += -L$$PATH_TO_LIBDATACHANNEL /Windows/Mingw64 -ldatachannel.dll
LIBS += -LC:/Qt/Tools/OpenSSLv3/Win_x64/bin -lcrypto-3-x64 -lssl-3-x64
LIBS += -lws2_32
LIBS += -lssp
```

1-3- نصب کتابخانه ی opus codec

حجم بسته های صوت بسته به bit rate و تعداد کانال ها و ... متغییر است اما به صورت کلی، ارسال بسته های حاوی محتویات صوتی به صورت خام (با کدک اصلی) کاری کمتر توصیه شده است زیرا حجم بالای انتقال داده ممکن است باعث کاهش کیفیت مکالمه ی زنده شود، به همین جهت یکی از بهترین کارها، استفاده از یک کدک به خصوص برای کاهش حجم بسته ها است. این کدک حجم بسته ها را به طرز چشمگیری کاهش میدهد و از طرفی کیفیت کاهش یافته و دقت از دست رفته در برابر حجم کاهش یافته قابل چشم پوشی ست. شما میتوانید با build کردن این کتابخانه بر روی سیستم خود و استفاده از آن برای encode کردن بسته های ارسالی به دریافت کننده و همچنین decode بسته های دریافتی از ارسال کننده، کیفیت بالای ارتباط خود را بر روی شبکه های ضعیف تر نیز حفظ کنید. نصب این کتابخانه بر روی ویندوز: (بر روی لینوکس نیست موارد نسبتا مشابه است):

```
git clone https://github.com/xiph/opus.git
cd opus
cmake -B Windows/Mingw64 -DCMAKE_BUILD_TYPE=Release -G "MinGW Makefiles"
cmake --build Windows/Mingw64
```

پس از build کردن این کتابخانه نوبت به لینک کردن آن در پروژه میرسد:

```
INCLUDEPATH += $$PATH_TO_OPUS/include

LIBS += -L$$PATH_TO_OPUS /Windows/Mingw64 -lopus
```

4-1- نصب Coturn

در ابتدا مراحل نصب و کانفیگ را طبق توضیحات ذکر شده در صفحه گیتهاب Coturn انجام دهید (همچنین بدین منظور می‌توانید از این لینک نیز استفاده کنید). توجه داشته باشید که در صورتی که تمایل به انجام این بخش امتیازی دارید، لازم است تا از این ابزار به عنوان Turn server و Stub server خود استفاده کنید. برای استفاده از این ابزار دو مسیر متفاوت را میتوناید استفاده کنید: الف) استفاده از docker ب) build && compile که هر دو روش نیز ساده هستند.

الف) استفاده از داکر یا نصب روی لینوکس: برای این موارد به گیت هاب رجوع کنید (بخش Installing / Getting started)

ب) build && compile: برای این منظور احتیاج دارید تا در ابتدا ابزار libevent2 را کامپایل کرده و در حین build کردن coturn مسیر باینری های آن را به make یا cmake بدهید.

https://github.com/libevent/libevent.git

نمونه ای از کانفیگ turn server برای coturn در فایل turnserver.conf در اختیار شما قرار میگیرد (فایل های پیوست تمرین)

نمونه ای از کانفیگ coturn در docker-compose در ابتدا صفحه بعد قابل مشاهده است:

```yaml
coturn:
  <<: *x-service-defaults
  image: coturn/coturn
  container_name: coturn-${DEPLOY_ENV}-x
  profiles:
    - dev
    - test
    - demo
    - prod
  # ports:
    # - "3030:3030"
    # - "3478:3478"
    # - "5349:5349"
    # - "5766:5766"
    # - "8080:8080"
    # - "49152-65535:49152-65535"
  environment:
    - COTURN_USERNAME=${COTURN_USERNAME}
    - COTURN_PASSWORD=${COTURN_PASSWORD}
    - COTURN_CLI_PASSWORD={COTURN_CLI_PASSWORD}
    - DOMAIN=${DOMAIN}
  restart: on-failure
  volumes:
    - ./coturn/turnserver.conf:/etc/coturn/turnserver.conf
    - /etc/letsencrypt:/etc/letsencrypt:ro
  networks:
    - proxy
  # healthcheck:
  #   test: ["CMD", "curl", "-f", "http://localhost:3030"]
  #   interval: 30s
  #   timeout: 10s
  #   retries: 3
  cap_drop: []
  cap_add:
    - SYS_PTRACE
    - CHOWN
    - SETGID
    - SETUID
```

Figure 1- نمونه کانفیگ

2- گام اول: نوشتن برنامه

2-1- رابط کاربری

یک رابط کاربری ساده جهت شبیه سازی تماس با موبایل پیاده سازی شده است (main.qml)، شما باید کلاس های مربوط را از CPP به رابط کاربری متصل نمایید تا بتوانید منطق برنامه را اجرا کنید.

2-2- کلاس WebRTC

یک Template در اختیار شما قرار میگیرد و لازم است تا آن را تکمیل نمایید، با تکمل توابع موجود در این کلاس، میتوانید از طریق این پروتکل اقدام به ارسال و دریافت بسته های صوتی نمایید. دقت کنید که استفاده از data channel ها برای ارسال و دریافت بسته های صوتی مجاز نیست و باید از ویژگی Track استفاده نمایید.

2-3- کلاس AudioInput

برای پیاده سازی این کلاس لازم است تا از QIODevice ارث بری کنید، سپس باید تابع

qint64 AudioInput::writeData(const char *data, qint64 len)

را پیاده سازی کنید، به محض آماده شدن بسته ی صوتی جدید، این بسته به این تابع ارسال خواهد شد. لازم است از opus encoder در این تابع استفاده کنید.

جهت شروع فرآیند دریافت شده از میکروفون نیز باید یک instance از کلاس QAudioSource در کلاس خود بسازید و تابع start را فراخوانی کنید.

2-4- کلاس AudioOutput

این کلاس پیچیدگی بیشتری دارد، برای پیاده سازی آن باید از QObject ارث بری کنید و از کلاس های QIODevice و QAudioFormat و QAudioSink و QMediaDevices و QMutex در آن instance بسازید.

با تابعی به اسم addData به محض رسیدن بسته ی جدید از مبدا آن را به یک صف اضافه کنید و با استفاده از سیگنالینگ سیستم Qt تابع play را فراخوانی کنید تا بسته ی اول صف را پخش کند. (پخش از طریق write کردن در QIODevice)

* در توابع addData و play لازم است تا از QMutexLocker استفاده شود.

* در addData سیگنال newPacket را emit کنید و در constructor این سیگنال را به تابع play وصل کنید.

* آبجکت مربوط به QIODevice را با استفاده از تابع start در QAudioSink مقدار دهی کنید.

* در تابع play از opus decoder استفاده کنید.

2-5- پیاده سازی Signaling Server

فرآیند اولیه ی اتصال بین دو peer در webrtc بدین شرح است که نفر اول به عنوان offerer یک SDP میسازد و این SDP باید از طریق یک سرور (شخص ثالث) به دست نفر دوم برسد، نفر دوم نیز یک SDP تحت عوان answerer میسازد و این نیز باید از طریق سرور به دست نفر اول برسد. پس از اینکه هر دو peer از SDP یکدیگر اگاه شدند، ارتباط مستقیم شکل میگیرد و برای ارسال بسته های صدا احتیاجی به سرور وجود نخواهیم داشت. ار طرفی، STUN سرور ICE Candidate هایی generate میکند که این دو نیز باید بین هر دو peer به اشتراک گذاشته شوند (به محض generate شدن یک ICE Candidate جدید باید آن را برای طرف مقابل ارسال کنیم)

وظیفه ی جایجایی SDP و remote ICE candidate ها و همچنین ذخیره ی caller ID ها بر عهده ی signaling server است. Signaling Server را میتوانید به هر زبان دلخواهی پیاده سازی کنید. برای این منظور پیشنهاد ما استفاده از فریمورک

Qt یا استفاده از nodejs است (دقت کنید در صورت استفاده از nodejs باید یک پیاده سازی شخص سوم از QtWebSocket برای nodejs را نصب کنید یا از Socket.io استفاده کنید) پیشنهاد ما استفاده از Socket.io است.

برای نصب و استفاده از socket.io در cpp کافی است تا مراحل زیر را انجام بدهید:

git clone https://github.com/socketio/socket.io-client-cpp.git

cd socket.io-client-cpp

git submodule update --init --recursive

Copying the contents of the src folder to the source code of your program in a folder named "SocketIO"

با انجام مرحه ی آخر، کدهای این کتابخانه همانند کد های خودتان در برنامه کامپایل خواهند شد، لازم به ذکر است که اضافه کردن فولدر SocketIO باید از QMake انجام شود:

```
HEADERS += \
   $$PWD/SocketIO/sio_client.h \
   $$PWD/SocketIO /sio_message.h \
   $$PWD/SocketIO /sio_socket.h \
   $$PWD/SocketIO /internal/sio_client_impl.h \
   $$PWD/SocketIO /internal /sio_packet.h
SOURCES += \
   $$PWD/SocketIO /sio_client.cpp \
   $$PWD/SocketIO /sio_socket.cpp \
   $$PWD/SocketIO /internal /sio_client_impl.cpp \
   $$PWD/SocketIO /internal /sio_packet.cpp

INCLUDEPATH += $$PATH_TO_SIO/lib/websocketpp
INCLUDEPATH += $$PATH_TO_SIO /lib/asio/asio/include
INCLUDEPATH += $$PATH_TO_SIO/lib/rapidjson/include

DEFINES += _WEBSOCKETPP_CPP11_STL_
DEFINES += _WEBSOCKETPP_CPP11_FUNCTIONAL_
DEFINES += SIO_TLS
```

6-2- کانفیگ و استفاده از coturn

مراجعه به بخش 4 – نصب Coturn

3- گام دوم

برنامه در پایه ای ترین حالت باید به گونه ای باشد که دو نسخه از آن روی یک لپ تاپ اجرا شود و صدا بین دو نسخه ی درحال اجرا منتقل شود، در حالت های پیچیده تر (امتیازی) دو دستگاه مختلف به یک دیگر داده ارسال میکنند. توجه شود که میتوانید callerID های مربوط به مخاطبان را در سیگنالینگ سرور hard code کنید. نیازی به دیتابیس یا فایل JSON نیست. از انجام هرگونه بافر و ارسال با تاخیر یا تجمیعی بسته ها خودداری کنید!

4- گام سوم

در این بخش نسبت به تکنولوژی WebRTC و Coturn اطلاعاتی جمع کرده و در فایل گزارشتان این اطلاعات را بیان کنید (مزایا، معایب، STUN, TURN و SIGNALING server و ...) از کپی پیست کردن اطلاعات جدا پرهیز کنید و صرفا آنچه از فرآیند های دخیل در این پروتکل متوجه شدید را ذکر کنید! مهم یادگیری و آشنایی شما با این پروتکل است نه حجم گزارش ارسالی!

انجام این بخش برای تمامی گروه ها اجباری است و ارتباطی با بخش های امتیازی ندارد!

5- جمع بندی و نکات پایانی

- مهلت تحویل: **1403/08/10**
- پروژه در گروه‌های 2 نفره انجام می‌شود. (گروه بندی در سامانه ایلرن نیز انجام می‌شود و تحویل تمرین به صورت گروهی خواهد بود)
- هر ۲ نفر می بایست کار را تقسیم کنند. همچنین از Git برای ساختن branch و تقسیم issue ها استفاده نمایید. (با استفاده از commit ها و تعیین issue ها میزان مشارکت هر نفر مشخص می‌شود). بعد از انجام این کار کدها را در یک repository به نام CN_CA_1 در اکانت‌های GitHub/GitLab خود قرار دهید(به صورت private). همچنین در یک فایل README.md می‌توانید report و داکیومنت خود را کامل کنید و در کنار repository قرار دهید. در نهایت لینک این repository را در محل پاسخ تمرین قرار دهید. (از فرستادن فایل به صورت زیپ جدا خودداری نمایید.) اکانت دستیاران این تمرین را به Repo خودتان به عنوان Maintainer به پروژه اضافه کنید.

Github accounts:

@TheSohrabX

@Hasti-Karimi
@mvali99

- برای پیاده سازی این تمرین از C++ استفاده کنید.
- دقت کنید گزارش نهایی شما می‌بایست همانند یک Document باشد و شامل توضیح کد و ساختار کد، همچنین نتیجه نهایی اجرای کد و اسکرین شات‌های دقیق از تمام مراحل باشد. (در فایل Readme.md کنار فایل های اصلی خود و در Repo مربوطه قرار دهید.) این نکته حائز اهمیت است که فایل PDF به هیچ عنوان مورد پذیرش قرار نخواهد گرفت ولی لازم است لینک Repository خود را در جایگاه تعریف شده برای این تمرین در ایلرن قرار دهید.
- ساختار صحیح و تمیزی کد برنامه، بخشی از نمره‌ی این پروژه شما خواهد بود. بنابراین در طراحی ساختار برنامه دقت به خرج دهید.
- برای هر قسمت کد، گزارش دقیق و شفاف بنویسید. کدهای ضمیمه شده بدون گزارش مربوطه نمره‌ای نخواهند داشت.
- هدف این تمرین یادگیری شماست. لطفا تمرین را خودتان انجام دهید. در صورت مشاهده‌ی مشابهت بین کدهای دو گروه، مطابقت سیاست درس با گروه متقلب و تقلب دهنده برخورد خواهد شد. همچنین توجه داشته باشید استفاده از ابزارهای AI توجیهی برای شباهت کدهای تحویل داده شده توسط گروه های مختلف نمی باشد.
- سؤالات خود را تا حد ممکن در گروه تلگرام درس مطرح کنید تا سایر دانشجویان نیز از پاسخ آن بهره‌مند شوند. در صورتی که قصد مطرح کردن سؤال خاص‌تری دارید، از طریق ایمیل زیر ارتباط برقرار کنید. توجه داشته باشید که سایر شبکه‌های اجتماعی راه ارتباطی رسمی با دستیاران آموزشی نیست و دستیاران آموزشی مؤظف به پاسخگویی در محیط‌های غیررسمی نیستند.
    - m.moradi1998@ut.ac.ir
    - hasti.karimi@ut.ac.ir
    - mvali@ut.ac.ir

موفق باشید