Spesifikasi Tugas Besar 2

IF3130 - Jaringan Komputer

“My Gacha Game Is So Unpopular, I Decided to Become a VTuber Instead”

Simple Audio Streaming Application

Dipersiapkan oleh:

Asisten Lab Sistem Terdistribusi

Didukung oleh:

Waktu Mulai :

Kamis, 12 November 2020, 20.21 WIB

Waktu Akhir :

Kamis, 26 November 2020, 23.55 WIB

I. Latar Belakang

Dua bulan telah berlalu sejak kamu memutuskan untuk membuat game *gacha*-mu sendiri. Dengan suatu keajaiban, kamu berhasil menyelesaikan game tersebut hanya dalam waktu 1 bulan. Pada bulan yang sama, game tersebut pun telah kamu rilis. Akan tetapi, sampai sekarang, jumlah *player* yang telah mengunduh dan memainkan game mu masih sama, yaitu 0.

Kamu pun menyadari kalau kemampuan ngoding yang dewa pun tidak cukup untuk menghasilkan uang dengan membuat game gacha. Kamu membutuhkan suatu teknik *marketing* agar orang-orang di luar sana bisa lebih mengetahui tentang game yang kamu buat. Apalah arti sebuah game *gacha* jika tidak menyebarkan ~~garam~~ keseruannya kepada orang lain.

Setelah sesi introspeksi yang memakan waktu beberapa jam, matamu pun terbuka dan kamu mendapatkan sebuah pencerahan. Rasanya seakan ada seseorang yang membisikkan jawaban atas segala kesulitan yang telah kamu alami hingga saat ini. Ya, jawabannya adalah...

“Aku akan menjadi seorang VTuber!”

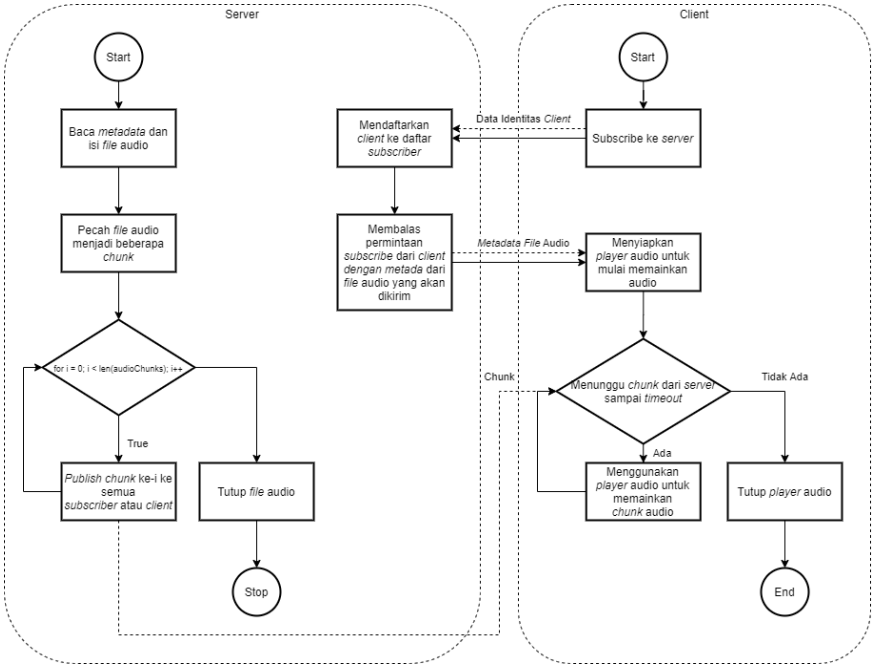
Kenapa VTuber? Untuk menjelaskan alasannya, maka kita perlu *flashback* ke 2 minggu yang lalu. Pada saat membuka sebuah situs video yang biasa kamu kunjungi, kamu direkomendasikan sebuah video yang memperlihatkan seorang VTuber bertelinga kelinci yang nampaknya dapat mengirim seseorang ke *isekai* dengan satu ayunan wortelnya. Kembali ke masa kini, kamu telah menjadi penggemar berat dari sebuah agensi VTuber dan kamu sudah percaya dengan kekuatan dari VTuber. Dari lubuk hatimu yang paling dalam kamu percaya kalau VTuber punya kekuatan untuk mengubah dunia.

Dengan tekad dan kepercayaanmu itu, kamu pun mengambil langkah pertamamu untuk menjadi seorang VTuber. Kamu berpikir bahwa menggunakan situs *streaming* yang biasa saja tidak cukup untuk membuatmu jadi populer karena sudah banyak VTuber di situs-situs tersebut. Kamu pun berencana untuk membuat situs *streaming* sendiri dengan kemampuan ngoding dewa yang kamu miliki. Namun, sebelum itu, kamu memutuskan untuk membuat sebuah proyek *streaming* kecil untuk meningkatkan pemahamanmu terkait *streaming* agar situs yang kamu buat nantinya dapat berjalan dengan baik dan kamu dapat akhirnya menjadi populer dan mendapatkan banyak uang.

II. Deskripsi Tugas

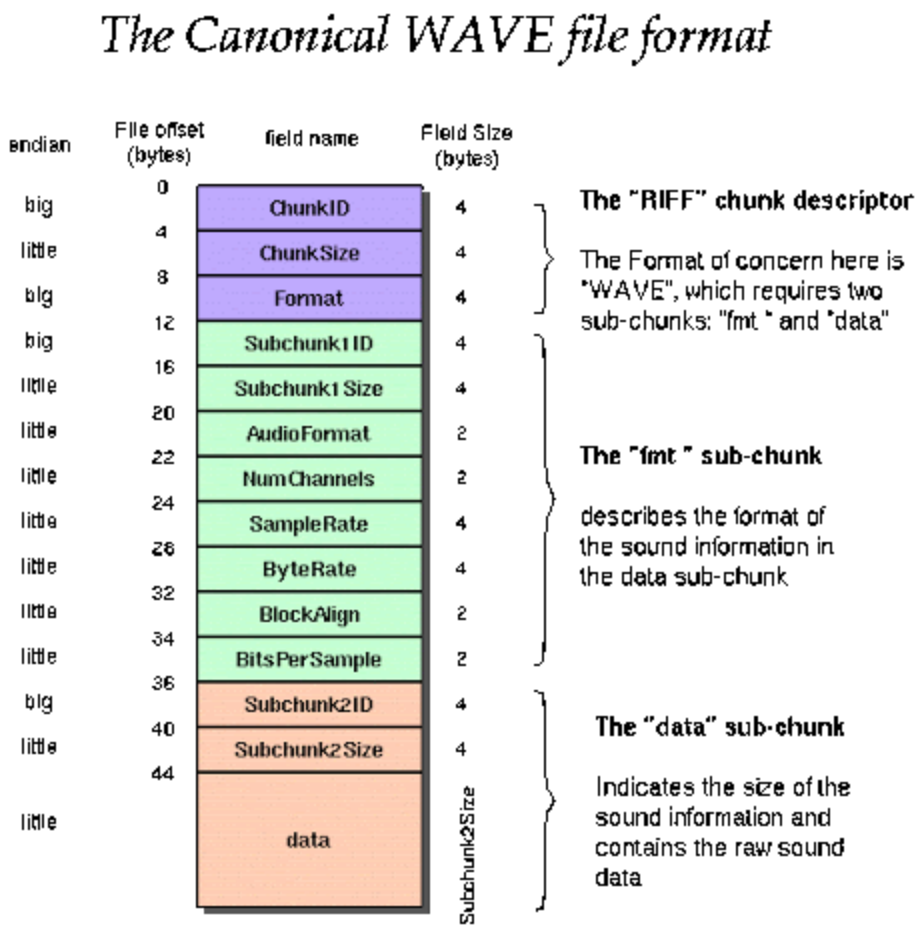
Dalam tugas besar ini, anda akan mengimplementasikan program yang akan melakukan pengiriman atau *streaming* suatu *file* audio dalam format WAV dan langsung memutarkan *file* audio tersebut. Anda dapat menggunakan program pada tugas besar anda sebelumnya untuk melakukan pengiriman *file* tersebut. Pastikan bahwa program anda dapat melakukan pengiriman *file* secara konkuren (hampir bersamaan) ke semua *client* yang terhubung. Jika belum, silahkan modifikasi program anda.

Terdapat dua buah program yang harus anda buat, yaitu *client* yang akan menerima pengiriman *file* audio dan memainkannya secara *real time*, dan *server* yang akan mengirimkan *file* audio tersebut ke semua *client* yang terhubung ke *server*. Proses pengiriman *file* audio atau *streaming* dari awal dapat dilihat melalui diagram berikut.



Gambar 1. Diagram proses *streaming*. (Pranala: https://drive.google.com/file/d/1PS3PKy545H_qdYWQfOUeCvkrszI0hYHP/view?usp=sharing)

Pada tugas ini, digunakan format *file* audio yaitu WAV. Adapun suatu *file* WAV memiliki struktur sebagai berikut.



Gambar 2. Struktur suatu *file* audio dengan format WAV.

Berdasarkan struktur tersebut, dapat dilihat bahwa secara garis besar, suatu *file* audio dengan format WAV dapat dibagi menjadi dua bagian, yaitu bagian *header* dan bagian *data*. Bagian *header* terletak di bagian awal *file* sampai *byte* ke-43 (panjang bagian *header* adalah 44 *byte*). Setelahnya, semua *byte* adalah *byte* data dari audio tersebut. Untuk mengekstrak *metadata* dari suatu *file* audio dengan format WAV, dapat diambil dari bagian *header* sesuai dengan struktur pada gambar. Selanjutnya, data dari *audio* berada di bagian *data*.

Sesuai dengan diagram proses *streaming* sebelumnya, maka program *server* harus mengekstrak *metadata* dari *file* audio terlebih dahulu sebelum melakukan

pengiriman data. Adapun untuk melakukan pengekstrakan *metadata* seperti *sample rate*, jumlah *channel*, *sample width*, dan *metadata* lainnya yang mungkin diperlukan, dianjurkan untuk menggunakan pustaka **wave** yang disediakan oleh **Python**. Namun anda tetap dapat melakukan ekstraksi *metadata* sendiri secara manual jika perlu. *Metadata* yang perlu dikirimkan ke *client* bervariasi tergantung pustaka yang digunakan oleh program *client* anda untuk memainkan audio, namun dianjurkan untuk menggunakan pustaka **PyAudio** untuk memainkan audio pada program *client* anda. *Metadata* yang diperlukan oleh **PyAudio** minimal mengandung *sample width*, jumlah *channel*, dan *sample rate*.

Setelah melakukan ekstraksi *metadata*, langkah selanjutnya adalah membagi bagian *data* dari *file* audio tersebut menjadi beberapa *chunk* yang kemudian akan di-*publish* satu per satu. Ukuran untuk satu *chunk* dibebaskan, namun harus tidak lebih besar dari ukuran maksimal satu *packet* UDP. Setelah membagi bagian *data* menjadi *chunk*, maka lakukan pengiriman tiap *chunk* tersebut ke semua *client* yang sudah melakukan *subscribe* ke *server* tersebut. Pengiriman tiap *chunk* tetap dilakukan meskipun tidak ada *client* yang terhubung, maksudnya adalah *chunk* tetap dianggap sudah terkirim dan lanjut ke *chunk* berikutnya (layaknya *streaming* pada umumnya). Setelah *chunk* berhasil dikirim ke semua *subscriber*, *server* wajib melakukan *sleep* selama waktu audio yang dikandung oleh satu *chunk*. Secara umum, berikut ini adalah *pseudocode* untuk *loop* pengiriman *chunk*.

```
CHUNK_SIZE = 32768 # For example, in bytes

nChannels = 2 # Audio channel count, just for example, get from the WAV metadata.
sampleWidth = 2 # Audio sample width, just for example, get from the WAV metadata.
frameRate = 44100 # Audio frame rate, just for example, get from the WAV metadata.

frameSize = nChannels * sampleWidth # In bytes
frameCountPerChunk = CHUNK_SIZE / frameSize

chunkTime = 1000 * frameCountPerChunk / frameRate # In milliseconds.

for chunk in chunks:
  startTime = getCurrentTimeInMilliseconds()
  for subscriber in subscribers:
    send(chunk, subscriber)
  endTime = getCurrentTimeInMilliseconds()
  delta = endTime - startTime
  if delta < chunkTime:
    sleepInMilliseconds(chunkTime - delta) # Sleep for the remaining time if there is any.
  # Continue to next chunk
```

Pseudocode untuk *loop* pengiriman *chunk* ke semua *subscriber*.

Setelah semua *chunk* dikirim, maka *server* akan menutup *file* yang sebelumnya dibuka dan menghentikan programnya sendiri.

Pada program *client*, dilakukan *subscribe* ke *server* terlebih dahulu. *Client* akan mendaftarkan diri ke *server* agar dapat menerima *publish* dari *server* tersebut. Pada proses pendaftaran diri tersebut, *client* perlu mengirimkan identitas diri terkait *client* tersebut, minimal terdiri atas alamat IP dari *client* tersebut pada jaringan yang digunakan. Selanjutnya, *server* akan mendaftarkan *client* tersebut ke daftar *subscriber*-nya, yang digunakan oleh *server* untuk mem-*publish chunk*. Setelah proses pendaftaran berhasil dilakukan, *server* harus membalas permintaan pendaftaran *client* dengan *metadata* terkait file *audio* yang sedang dimainkan oleh *server*.

Setelah *client* menerima balasan dari *server* yang berupa *metadata* dari file audio, maka *client* akan mempersiapkan *player* audio berdasarkan *metadata* yang diterima tersebut.

Selanjutnya, program *client* akan melakukan *busy waiting*, menunggu *chunk* yang dikirimkan oleh *server* sampai *timeout* tercapai. *Timeout* dapat ditentukan berdasarkan waktu yang telah lewat dari waktu terakhir *client* menerima *chunk*, misal 10 detik, atau metode lain yang menurut anda cocok untuk menandakan telah terjadi *timeout*. Untuk setiap *chunk* yang diterima, maka *client* akan memainkan *chunk* audio tersebut. Jika *client* menerima *chunk* ketika sedang memainkan *chunk* sebelumnya, maka *chunk* baru tersebut dapat disimpan dahulu di RAM atau menggunakan metode lainnya, misal dengan *queue* pada RAM, yang selanjutnya akan dimainkan ketika *chunk* sebelumnya telah selesai dimainkan. Program *client* akan melakukan *busy waiting* ini sampai *timeout*, dan kemudian akan menutup program *client* tersebut.

Anda perlu menentukan struktur data pada *packet* anda, sesuai dengan kebutuhan program anda. Beberapa struktur data yang mungkin anda perlukan adalah struktur untuk identitas dari *client* yang akan dikirimkan oleh *client* dan struktur *metadata* yang dikirimkan oleh *server*. Jika tugas besar anda sebelumnya sudah bekerja dengan benar, seharusnya anda tidak perlu mengubah struktur

packet anda tersebut. Anda hanya perlu menentukan struktur data yang dikandung oleh *packet* anda tersebut, sesuai kebutuhan sebelumnya.

Untuk referensi lebih lanjut, anda dapat mengakses pranala-pranala berikut ini.

1. http://tiny.systems/software/soundProgrammer/WavFormatDocs.pdf
2. http://people.csail.mit.edu/hubert/pyaudio/
3. https://docs.python.org/3/library/wave.html

Untuk mempermudah pembuatan program anda, silahkan lihat contoh program sederhana di bawah ini. Program ini membuat beberapa *chunk* audio secara acak, kemudian memainkan setiap *chunk* tersebut (yang berupa *bytes*).

```
import random
import pyaudio

def main():
    # This upper part is not so relevant to your program, just to generate random audio.

    CHUNK_SIZE = 32768 # In bytes.

    sampleWidth = 2 # Just for example, get from metadata.
    nChannels = 2 # Just for example, get from metadata.
    frameRate = 44100 # Just for example, get from metadata.
    frameSize = sampleWidth * nChannels

    chunkFrameCount = CHUNK_SIZE / frameSize
    chunkTime = chunkFrameCount / frameRate # Time elapsed in one chunk. In seconds.

    # Get the number of chunks needed to create 3 seconds audio bytes.
    chunkCount = 0
    while chunkCount * chunkTime < 3:
        chunkCount += 1

    # Generate some random chunks.
    randomChunks = []
    for _ in range(chunkCount):
        randomChunk = bytearray()
        for _ in range(CHUNK_SIZE):
            randomBits = random.getrandbits(8)
            randomByte = randomBits.to_bytes(1, 'little')
            randomChunk += randomByte
        randomChunks.append(randomChunk)

    # This part below is relevant to your program.

    # Prepare the audio player.
    pAudio = pyaudio.PyAudio()
    stream = pAudio.open(
        format=pAudio.get_format_from_width(sampleWidth),
        channels=nChannels,
        rate=frameRate,
        output=True,
```

```
    )

    # Play each chunk separately.
    # In your programs, this will be done every time your program
    # receives a chunk from the server, instead of looping through
    # chunks.
    for chunk in randomChunks:
        stream.write(bytes(chunk))

if __name__ == '__main__':
    main()
```

Penggunaan

1. *Server* dijalankan dengan argumen input berupa *port* yang akan di-*bind* dan nama *file* audio berformat WAV yang akan di-*publish*.

2. *Client* dijalankan dengan argumen berupa alamat IP dari *server* yang ingin di-*subscribe*.

3. *Server* akan mem-*publish chunk* dari *file* audio tersebut kepada semua *subscriber* secara bersamaan (konkuren).

4. *Client* akan menunggu *chunk* dari *server* dan memainkan tiap *chunk* yang diterima secara *real time*.

Requirements

1. *Server* dapat mengirimkan *file* ke beberapa *client* secara konkuren.

2. *Client* dapat menerima dan memainkan *chunk* dari *file* audio yang dikirimkan oleh *server*.

3. *Sender* dan *receiver* dapat berada di komputer yang berbeda yang berada di jaringan yang sama.

4. Bahasa yang digunakan adalah **Python 3.** Kesalahan menggunakan bahasa akan menyebabkan tugas tidak dinilai.

5. Dilarang menggunakan pustaka yang bukan bawaan **Python 3** untuk melakukan pengiriman data. Anda boleh menggunakan pustaka untuk membaca dan memainkan *file* audio. Dianjurkan menggunakan pustaka **wave** untuk membaca bagian *metadata* dan *data* dari *file* WAV dan **PyAudio** untuk memainkan *bytes* dari *chunk* yang diterima *client* sesuai format WAV.

Bonus

1. GUI untuk menampilkan *metadata* dari *file* audio yang sedang di-*streaming* oleh *client*, misal nama *file*, menit dan detik yang sedang dimainkan, lama *file* audio, dan lainnya.
2. Dapat melakukan *rewind* pada *client*. *Chunk* boleh disimpan di RAM dengan metode yang anda tentukan sendiri.
3. *Autodiscover server*. *Client* dapat mengetahui alamat IP dari *server* yang berada di jaringan yang sama secara otomatis. Hal ini dapat dilakukan dengan *client* yang melakukan *query* dengan *flooding* atau *server* yang melakukan *announcement* dengan *flooding*. Anda dapat menambahkan struktur data sendiri untuk mendukung fitur ini.
4. Pengaturan kualitas *streaming*. *Client* dapat menentukan kualitas audio yang ingin di-*stream* ketika *subscribe*. *Server* akan mengirimkan *chunk* audio sesuai kualitas yang dipilih tersebut ke *client* terkait. Sebagai contoh, seperti kualitas video di YouTube (480p, 720p, 1080p, dan seterusnya).

Tambahan

1. Anda boleh menggunakan kode dari tugas besar sebelumnya milik teman anda yang sudah berhasil.
2. Tugas akan dikerjakan di *repository* pada Github Classroom dan dikumpulkan melalui Github Classroom.
3. Tidak ada *autograder* pada tugas kali ini. Penilaian akan dilakukan melalui demo program dengan asisten.
4. Waktu pengerjaan tugas besar ini adalah dua minggu sejak rilis tugas.

III. Pengumpulan dan Deliverables

1. Tiap kelompok diwajibkan untuk *join* ke Github Classroom yang akan diumumkan nanti melalui milis.
2. Ketika akan *join*, akan ditanyakan anggota kelompok. Silahkan isi anggota kelompok sesuai kelompok anda.
3. Setelah berhasil *join*, Github akan membuatkan repo untuk kelompok anda. Silahkan kerjakan tugas ini pada repositori tersebut.
4. Untuk men-*submit* tugas kelompok anda, gunakan *tag* `v1.0.0`.
5. Dilarang memodifikasi folder `.github`.