Dağıtık Sistemler ve Uygulamalar

Proje Raporu

Grup 5

Bu projenin amacı birden fazla peer'ın birbiri ile konuşması ve birbirleriyle dosya paylaşımı yapabilmeleridir. Öncelikle projenin gerçekleşebilmesi için aptal sunucu tasarlanmıştır. Bu sunucu sadece kendisine gelen bağlantı isteklerini kabul etmekte ve kendisine liste isteği gönderenlere, kendisinde kayıtlı olan kullanıcıları göndermektedir. Protokolümüzdeki **USR**, **LSQ**, **TIC** ve **QUI** komutlarına cevap döndürmektedir.

Komut	Parametre	Cevap	Parametre	Tanım
USR	uuid, IP, Port	HEL	uuid, P/S, timestamp	Yeni kullanıcı kabulü
		REJ	uuid	Yeni kullanıcı reddi
SRC	dosya_ismi	VAR	dosya_ismi, boyutu, md5 :: dosya_ismi, boyutu, md5 :: ...	Aranan dosya isimleriyle birlikte benzerlerini de listeler.
		YOK		Hiçbir dosya bulunamadı.

Sunucumuzda **LoggerThread** ve **ReaderThread** bulunmaktadır. LoggerThread, lQueue kuyruğuna server da gerçekleşen bütün işlemleri saat, thread numaraları, gönderilen ve gelen komutlarla birlikte kaydeder. ReaderThread de kendisine gelen komutları parse etmekte ve kendisine gelen komutlara protokole uygun olarak cevap yollamaktadır. MainThread de ise server sürekli olarak bağlantı beklemektedir.

```
class LoggerThread(threading.Thread):
    def __init__(self, name, logQueue, AptalSunucuLog):

    def log(self, message):

    def run(self):

class ReaderThread(threading.Thread):
    def __init__(self, name, csoc, fihrist, sendQueue, logQueue):

    def run(self):

    def parser(self, data):

while True:
    c, addr = s.accept()
    readerThread = ReaderThread("Reader Thread - "+ str(rCounter),
                                c, fihrist, sendQueue,
                                lQueue)
    readerThread.start()
```

Projedeki bir sonraki aşamamız, peer'ın backend'ini gerçeklemek oldu. Peer'da da öncelikle gerçekleşen bütün işlemlerin kaydını tutumak amacıyla bir LoggerThread oluşturuldu. Peer hem client hem de server olarak çalışacağı için her iki bölüm için de ayrı ayrı threadler oluşturuldu.

Server tarafını ele alıcak olursak, peer'ın server'ının sürekli accept() halinde kalmasıyla main thread'I bekleteceği için ayrıca bir server_starter thread oluşturuldu ve bu threadle birlikte gelen kullanıcıların bağlantı isteği kabul edildi, böylelikle main thread'e paralel olarak çalışması sağlandı.

```
class server_starter(threading.Thread):
    def __init__(self):
        threading.Thread.__init__(self)

    def run(self):
        fihrist = dict()
        lQueue = queue.Queue()
        lThread = loggerThread("Logger", lQueue, "log_client.txt")
        lThread.start()
        server_queue = queue.Queue(20)
        s = socket.socket()
        host_serv = "0.0.0.0"
        port_serv = PORT
        s.bind((host_serv, port_serv))
        s.listen(5)
        serverCounter = 1
        while True:
            c, addr = s.accept()
            print("Got connection from " + str(addr))
            serv_thr = server_thread("Server Thread" + str(serverCounter), c,
fihrist, server_queue, lQueue)
            serv_thr.start()
            serverCounter += 1
```

server_starter thread, **server_thread**'i çağırmaktadır. Bu thread sunucumuzdaki parser fonksiyonun benzer bir şekilde çalışmaktadır. Fark olarak dosya işlemleri için gerekli olan arama komutlarını içermektedir.

```
if data[0:3] == "SRC":

    def benzerlik(first_file, second_file):

    def search_files(filename):

    def get_md5(filename):

    def search_md5(choosen_md5):
```

SRC komutumuzun içerisinde benzerlik, search_file, get_md5, search_md5 metodlarımız bulunmaktadır.

- **benzerlik metodu;** parameter olarak iki tane dosya ismini alarak benzerlik karşılaştırmasını yapar.
- **search_files metodu;** benzerlik metodunun döndürdüğü benzerlik oranına göre dosyaların md5, dosya boyutu ve oluşturulma tarihini çekerek, en çok benzeyene göre sıralar.

- **get_md5 metodu;** dosyanın md5'ini döndürür.
- **search_md5 metodu;** seçilmiş olan dosyanın md5 değerine göre arama yapar.

Client tarafı için üç tane thread oluşturulmuştur. Bunlar; **readerThread**, **senderThread** ve **clienthandlerThread**'dir. readerThread ve senderThreadler bir peer'ın kullanacağı okuma ve yazma işlemlerini yapan threadler olarak düşünülebilir. Fakat bu projedeki amacımız aynı anda birden fazla peer ile bağlantı kurabilmek olduğu için bunlara ek olarak bir de clienthandlerThread'I oluşturulmuştur. Bu sayede de her bir bağlantı isteği için yeni bir soket açan bir thread oluşturulmuştur.

```
class readerThread (threading.Thread):

    def __init__(self, name, csoc, host, senderQueue):

    def incoming_parser(self, data):

    def run(self):

class senderThread (threading.Thread):

    def __init__(self, name, csoc, host, threadQueue):

    def run(self):

    def out_going(self, data):

class clienthandlerThread(threading.Thread):

    def __init__(self, name, host, port, queue, fileName):

    def run(self):
```

ClientHandlerThread, readerThread ve senderThread'ı çağırmaktadır. Her bir yeni bağlantı isteği için yeni bir soket ve kuyruk açmakta, bu soket ve kuyruğu ise parameter olarak senderThread ve readerThread'e vermektedir.Ayrıca arayüzle beraber clientHandler threade bazı eklemeler yapılmıştır bu arayüz kısmı anlatılırken eklenicektir.

SenderThreadın içerisinde **outgoing_parser** metodu bulunmaktadır.Bu metod gönderilecek mesajın (USR, TIC, SRC…) parselenmesi içindir. SenderThread kendi kuyruğundan aldığı inputu sokete koymakta ve clientın bağlandığı peera bu isteği yollamaktadır.

ReaderThread içerisinde **incoming_parser** metodu bulunmaktadır. Bu metod peer'ın client tarafının bağlandığı peer'ın sunucusundan dönen mesajı (HEL, TOC, VAR…) parse etmekte ve bu mesajı bastırmaktadır.

```
class readerThread (threading.Thread):

    def __init__(self, name, csoc, host, senderQueue):
        threading.Thread.__init__(self)
        self.name = name
        self.csoc = csoc
        self.senderQueue = senderQueue
        self.host = host
        self.port = str(PORT)


    def run(self):
        global file_dict
        while True:
            data = self.csoc.recv(1024)
            self.message = self.incoming_parser(data.decode())
            if self.message==-1:
                break
        self.csoc.close()

class senderThread (threading.Thread):

    def __init__(self, name, csoc, host, threadQueue):
        threading.Thread.__init__(self)
        self.name = name
        self.csoc = csoc
        self.senderQueue = threadQueue
        self.host = host
        self.port = str(PORT)

    def run(self):
        while True:

            if not self.senderQueue.empty():
                self.queue_message = self.senderQueue.get()
                self.out_going(self.queue_message)
```

Arayüz tasarımı Qt Designer ile yapılmıştır. Çıkan .ui uzantılı dosya .py uzantılı dosyaya convert edilip peerın backend ile beraber çalışıcak duruma getirilmeye çalışılmıştır.

```
class Ui_MainWindow (object):
    def setupUi(self, MainWindow):

    def retranslateUi(self, MainWindow):

    @pyqtSlot()
    def on_click(self):

    @pyqtSlot()
    def click_search(self):

    @pyqtSlot()
    def click(self):
```

Class Ui_MainWindow içerisinde Qt designer ile yapmış olduğumuz arayüzün kodu bulunmaktadır. Bunlara ek olarak **on_click**, **click_search** ve **click** fonksiyonları vardır. **On_click fonksiyonu** arayüzde bulunan connect butonuna basıldığı zaman yeni bir soket ve queue açmakta bu soket ile queue senderThread ve readerThread parameter olarak vermektedir. Ayrıca queue'ya "USR" isteği koymakta ve peerın ilk defa sisteme kayıt olması aşamasının gerçekleştirilmesini sağlamaktadır.

Click fonksiyonu ise ip ve portunu yazdığı peer ya da aptal sunucunun kendisine bağlı olan peerların bulunduğu listeyi çekmekte ve bu listede olan kişilere "USR" ve "SRC" isteği yollamaktadır.ClientHandler metoduna arayüz aşamasında ekleme yapılmış ve queue'lara bu komutlar konulmuştur.

Click_search fonksiyonu ise bağlantı kurduğu peerlardan yollanmış olan dosya isimlerini bastırmak için yazılmış bir metottur.

Hedeflenenler:

Click_search fonksiyonuna bastıktan sonra dosyalar listelenir ve seçilen dosyanın üstüne tıklandıktan sonra **search_md5** fonksiyonuna seçilmiş olan md5 değeri parametre olarak verilir. İndirilmek istenen md5 kesinleştikten sonra asıl koda import edilemeyen **fileserver.py** ve **fileClient.py** dosyalar yardımıyla indirme işlemi gerçekleştirilir.