INF 443

Dağıtık Sistemler ve Uygulamaları

[2015-2016]

P2P GÖRÜNTÜ İŞLEME UYGULAMASI -PROJE RAPORU-

Sümeyye KONAK

14401676

1. Projeyi Yaparken Geçilen Aşamalar

1.1. Filtreler

İlk olarak ip.py dosyasında var olan filtreleri inceledim. İstenen filtreler hakkında wikipedia gibi sitelerde yaptığım araştırmalar sonucu Binarize, Prewitt ve Gaussian filtrelerini yazdım. İşleri threadlerin bölüştüğü bir sisteme (ip.py) eklendim bu filtreleri. Çalışıp çalışmadıklarını ise ip.py'i çalıştırarak var olan bir arayüz (pyGraphics_ui.py) sayesinde test ettim. Gerekli düzeltmeleri yaptım.

1.2. Negotiator.py

Filtreler yazıldıktan sonra işin asıl kısmı olan 'dağıtık' karşıma çıktı. Projede var olan sistemin genel mimarisini anlama kısmı ise ilk yapılması gereken oldu. Genel mimariyi biraz anladıktan sonra negotiator hakkında daha ayrıntılı düşünmeye başladım. Daha önce yaptığımız ödevler sayesinde negotiator'da yaratmam gereken client ve server ana threadleri kafamda biraz daha netleşti. Policy'leri kağıt üzerinde belirledim. Sonuç olarak 2 ana thread ve o threadlerin içinde yarattığım iki yardımcı thread ile negotiator'ı yazım. Basit bir client.py yazarak negotiator'a REGME komutunu parametreleriyle gönderip CONNECT_POINT_LIST ve üzerinde testler yaptım. Son olarak yapılanları düzgün bir şekilde takip edebilmek adına bir log thread tanımladım.

1.3. Peer.py

Negotiator bittikten sonra onun biraz daha genişletilmiş (görüntü işleme özelliği gibi özelliklerin eklendiği) hali olarak kabul edebileceğimiz peer.py'yi yazmaya başladım. Öncelikle bu peer, negotiator'ın yaptığı her şeyi yaptığı için taslak olarak negotiator.py'yi aldım. Protokoldeki ihtiyaçlara göre gerekli threadleri ve metotları yazdım. Görüntü işleme aksiyonunu nasıl entegre edeceğimi bize verilen zaman diliminde bulamadığım için bu noktada bir test yapmam mümkün olmadı. Fakat peer ve negotiator arası bağlantıların düzgün bir şekilde gerçekleşmesi ve CONNECT_POINT_LIST güncellemeleri gibi durumları test ettim.

2. Dağıtık İşleme Senaryosu

Projenin bu aşamasını tam olarak gerçekleyememekle birlikte, projeden yola çıkarak yazacak olursam.

- Son kullanıcı açısından diğer tüm dağıtık sistemlerde olduğu gibi tek ve düzgün çalışan bir sistemin olması gerekiyordu. Yani proje sırasında karşılaştığımız sorunların son kullanıcıya yansıtılmadan tarafımızca tüm olası case'lerin ele alınması gerekiyordu. Daha net olacak olursam, örneğin bir parça işlenmek üzere A peer'ına gönderilmişse ve A peer'ı işleyip gönderemeden bağlantısını kapatmışsa, o halde bu parça başka bir peer'a gönderilip son kullanıcı bu çıkan sorunlardan

haberdar etmemek gerekiyordu. Bu noktada görüntü işlemeyi entegre edemediğimden bu gibi senaryolar üzerinde çalışma imkanı bulamadım.

- Son kullanıcının kullandığı eş açısından ise bu eş tıpkı diğer eşler gibi sisteme kayıt olur. Kendi peer listesini hem düzenli olarak günceller hem de kendi listesinde bulunan peer'ların listelerinde bulunanları da listesine alır. Bahsi geçen iki iş için peer.py'de gerekli threadler yaratıldı. Böylelikle tüm dağıtık sisteme hakim olmuş olur. Bu da işini çok daha hızlı ve kolay bir şekilde yapmasını sağlar.
- Fonksiyonu sağlayan eş kendisine gelen işleme işini yapıp, ilgili eşe gönderir. Bu konuda fonksiyonlarla ilintili bir test yapamadım.
- Arabulucu açısından ise durumun biraz daha basit olmasına karşın dikkat gerektirir. Bu noktada asıl işi CONNECT_POINT_LIST'in her zaman en son doğru olan veriyi tutmasını sağlamaktır. Bunu da client tarafından düzenli olarak mesajlar göndererek ve gelen mesajları hızlı ve düzgün bir şekilde parse edip bünyesine alarak yapar.

3. İki Tane Çözümü Bulunamamış Problem

3.1. Patchsize Problemi

Aslında bu problem ip.py'yi run ettiğimizde ilk olarak karşımıza çıktı. Orada da dağıtık sistemimizde olduğu gibi patchsize sabitti ve patchsize'ın sabit olduğu bir sistemde eğer load edilen görüntü patchsize'a tam bölünmüyorsa görüntünün bazı kısımları işlenmeden kalıyor. Çözüm olarak arayüzde kullanıcıyı load edebilmesi için belirli görüntü size'ları verip bunun haricinde olanları kabul etmeyebiliriz ya da yüklenen görüntüye göre patchsize'ı belirleyebiliriz.

4. Uygulama Protokolünde Yapılabilecek Değişiklikler

4.1. FUNLS'in kaldırılması

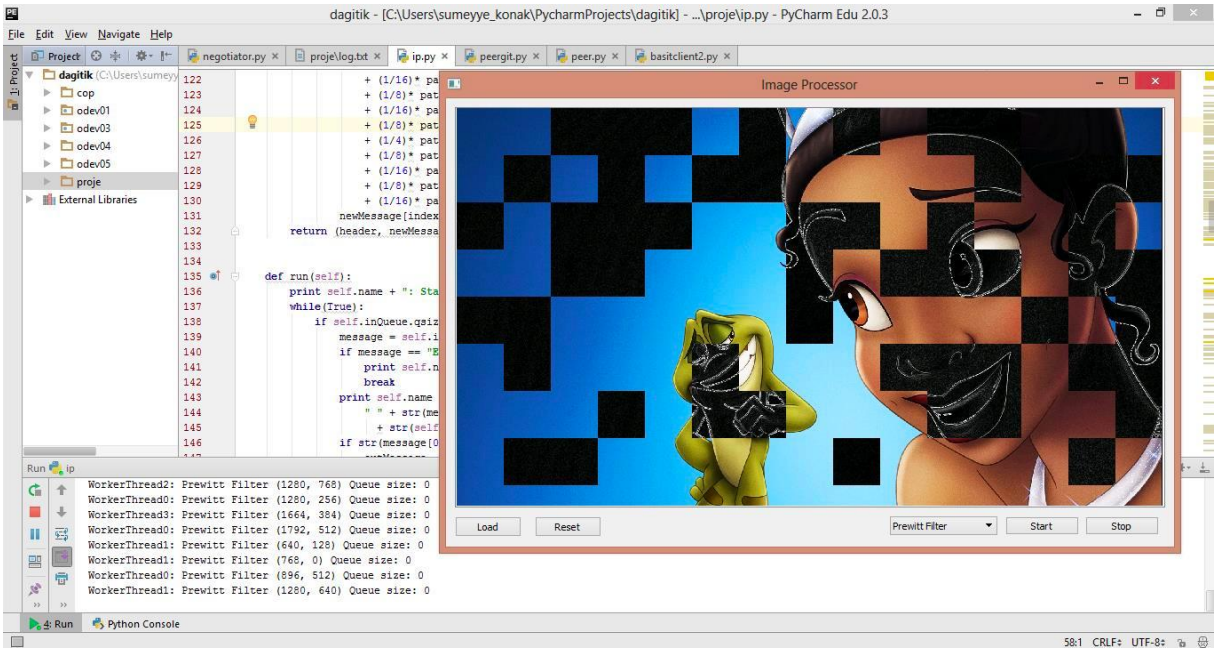
FUNRQ sorgusu olduğu için bunun öncesine FUNLS komutunun göndermenin bir anlamı yok. Bu komut protokolden kaldırılabilir.

4.2. <type> bilgisinin kaldırılması

Her bir negotiator'ı aslında görüntü işleyemeyen birer peer olarak düşünebiliriz. Böyle baktığımızda aslında protokol mesajlarında geçen <type> bilgisinin herhangi bir yaptırımı yok. Her peer zaten başlangıçta bir negotiator'ın ip/port'unu bilmek durumunda. Buna ek olarak her seferinde özellikle de CONNECT_POINT_LIST'te bu bilgiyi tutmamıza gerek olmadığını düşünüyorum. Kodu yazarken başlangıçta <type> bilgisini tuttum fakat gerçekten ihtiyacımın olduğu bir durumla karşılaşmadım.

5. Ekran Çıktıları

5.1. Dağıtık Olmayan Bir Sistemde Filtrelerin Testi



5.2. Negotiator'da Bulunan log.txt Çıktıları

(deneme için UPDATE_INTERVAL'i 20 saniye olarak seçmiştim.)