# http://upload.wikimedia.org/wikipedia/en/9/96/Sakarya_%C3%9Cniversitesi_(logo).jpg

# SAKARYA ÜNİVERSİTESİ

# BİLGİSAYAR VE BİLİŞİM BİLİMLERİ FAKÜLTESİ

# BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

# İŞLETİM SİSTEMLERİ

# Proje

|  |  |
| --- | --- |
| **Ad Soyadı** | **No** |
| Seidy Kante | B191210559 |
| Bedru Umer Mohammed | B191210557 |
| Caner AY | B211210302 |
| Alperen Oruç | B201210047 |
| Doğukan Göktaş | B211210301 |

Github Repo link: <https://github.com/bedre7/dispatcher-shell>

**Giriş**

Günümüzdeki işletim sistemlerin oldukça karmaşık ve gerçek hayatta var olan proses yönetme problemi, olabilecek en iyi algoritma ile çözmektedir. Bilgisayarda binlerce proseslerin aynı anda çalışıyormuş gibi algılansa da arka planda etkin bir şekilde çalışan *zamanlayıcı* algoritmalar bulunmaktadır.

Proseslerin zamanında bitirilmesi için süreçlerin/işlerin çizelgelenmesi yapılır. CPU Zamanlayıcı, bir prosesin CPU'yu kullanmasına izin verirken başka bir prosesin I/O vb. herhangi bir kaynağın bulunmaması nedeniyle gecikmeli (beklemede) olması ve böylece CPU'nun mümkün olduğu kadar kullanılması işlemidir. CPU Çizelgelemenin amacı, sistemi daha verimli, daha hızlı ve daha adil hale getirmektir.

**Program işleyişi**

**👉** Bu ödevde prosesleri yönetmek için **Round Robin(Geri beslemeli)** ve **First-Come-First-Served (FCFS)** zamanlama algoritmaları kullanılmıştır.

**👉 Dispatcher(Dağıtıcı)** sınıfının görevleri – parametreden verilen dosyayı okuyup proses nesnelerini oluşturmak; proseslerin öncelikleri ve geliş zamanlarına göre uygun kuyruğa atamak ve de çalışması gereken kuyruğu çalıştırmaktır.

**👉** Prosesleri dosyadan geldiği gibi çalıştırmak yerine önceliklerine göre sıralanmıştır. Sıralama için *java.util’den* **PriorityQueue** kullanılmıştır. Daha sonrasında özel sıralama parametrelerimiz olacağı için java’ya ait **Comparator** adlı sınıfından miras alarak kendimiz özgü bir proses karşılaştırıcı (**ProcessComparator**) sınıfı sıralama koşulları belirledik.

Bu koşullar ise, ilk olarak önceliklerine göre karşılaştırılmaktadır. Öncelikleri eşit olması durumunda ise geliş zamanlarına bakılmakta.

**👉**  Sıralanan prosesleri daha esnek bir şekilde gezebilmek için kuyruktan yığına aktarılmıştır.

**👉 Dağıtıcının start()** fonksiyonun aracılığıyla programımız başlatılıyor. Bu fonksiyonun içinde her saniyede gelen proses varsa ilk olarak gerçek zamanlı olup olmadığını kontrol edilir, Gerçek zamanlı(Real Time process) ise kesintisiz FCFS algoritmasına göre çalışan kuyruğa atılır. Aksi halde, ilgili önceliğine göre uygun kuyruğa aktarılır.

**👉** Her saniyede gelen proses, son çalıştırılan prosesten daha öncelikli ise son proses askıda alınmıştır(**Interrupt**). Askıda alınan prosesler bekletilir.

**👉** Bekleyen proseslerin 20 saniyeden aşıp aşmadıklarını takip edebilmek için bir başka bekleyen proseslerin listesinde tutulur.

**👉** Prosesler çalıştırlırken **Timer** bir artırılır ve prosesin durumu(**State**) güncellenir.

**👉 Timer** artırılırken 20sn bekleme zamanı aşan proses varsa Dispatcher tarafından sonlandırılır.

**👉 RealTimeQueue** yukarıda bahsedildiği gibi gerçek zamanlı prosesleri kesintisiz(FCFS) bir şekilde çalışan proseslerin kuyruğu barındıran bir sınıftır.

**👉** **UserJob(Dağıtıcı)** uç adet proses kuyruğunu barındıran geri beslemeli Round Robin (kesintili) algoritmasına sahip ve proseslerin önceliğine göre dağıtan bir sınıftır. Bu sınıfın içinde gelen prosesin önceliğini kontrol edip ilgili kuyruğa atılır. Dispatcher tarafından çalışmasına izin verilirse en öncelikli kuyruktan sıradaki prosesi 1sn kuantuma kadar çalıştırıp önceliğini düşürerek bir sonraki kuyruğa aktarılır.

**👉** Böylece işlem her saniyede yürütülmektedir.

**Öğrendiklerim**