**1fd75c3872a94e26ad68c7fa7667bdc82c07dd4ba85f4a6793f7a2b4e943b8fd**

**T.C.**

**SAKARYA ÜNİVERSİTESİ**

**BİLGİSAYAR VE BİLİŞİM BİLİMLERİ FAKÜLTESİ**

**BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ**

İŞLETİM SİSTEMELERİ PROJE RAPORU

**İŞLETİM SİSTEMLERİ**

Proje için : <https://github.com/EreenKara/OperatingSystem>

**B211210031 - Eren KARA - 1/C**

**B211210047 Emirhan ETLİ - 1/A**

**B221210379 Mervan AKSU - 1/C**

**B…….. Samed DEĞER - 1/B**

**B211210101 İlyas AYDIN - 1/A**

**SAKARYA**

**Aralık, 2023**

İçindekiler:

1. Proje Özeti
2. Uygulama Tasarım Modeli
3. Proje Tasarımının Tartışılması
   1. Neden Böyle Bir Tasarım Tercih Ettik
   2. Interface İle Çalışma
   3. Proje Dependency İlişkisi
   4. Zaman Modülü
   5. Observer Deseni İle Asenkron Gerçekleme
   6. Bellek Algoritmamızın Seçilme Sebebi
   7. Görevlendirici Yapısının Kurulması ve Olası Başka Tasarımlar
   8. Scheduler Algoritmalarının Tartışılması
4. Proje Akış Mantığı
5. Uygulamaya Ait Görseller
6. Proje Özeti  
    Projede bize verilmiş kurallarla gerçeklenen bir işletim sistemi tasarlamaya çalıştık. Bunu gerçeğe en yakın bir şekilde yapmak için uğraştık ve bu yüzden projenin ana şemasının dışında yardımcı birden fazla sınıf, çalışma mantığı gerçekledik. Algoritmamızı temel bir zaman modülünün çalışması üzerine kurguladık ve bunun için kendi zamanımızı tasarladık. Bu zaman dilimleri arasında ise asıl algoritma gerçeklenecek. Detaylı açıklama 4. bölüm akış mantığı içerisinde yer almakta.  
   Proje içerisindeki dokümanlara ulaşmak için : <https://github.com/EreenKara/OperatingSystem/tree/master/docs>
7. metin, ekran görüntüsü, diyagram içeren bir resim

   Açıklama otomatik olarak oluşturulduUygulama Tasarım Modeli  
   Tasarım modeline ulaşmak için : <https://github.com/EreenKara/OperatingSystem/blob/master/docs/OperatingSystems.drawio>
8. Proje Tasarımının Tartışılması
   1. Neden Böyle Bir Tasarım Tercih Ettik ?

Tasarımımızı internet araştırmalarımızda elde ettiğimiz gerçek işletim sistemi çalışma mekanizmalarına göre gerçekledik. Proje içerisindeki her bir field, her bir algoritma gerçekte hangi modül (cihaz) tarafından gerçekleniyorsa bizim işletim sistemimizde de aynı modül tarafından gerçekleniyor. Gerçekçiliğin artması ve algoritmanın akışının daha anlaşılabilir olması açısından birçok sınıf ilave ettik. Örneğin process’lerin anlamsız bir sınıf tarafından sıralama algoritmalarına gönderilmesinden daha ziyade gerçek bir kullanıcı tarafından veriliyormuşcasına User sınıfı tarafından Operating System’in kontrolünü gerçekledik. Buna benzer gerçekçiliği arttırıcı birçok detay bulunmakta.

* 1. Interface İle Çalışma

Proje tasarımını gerçeklerken bütün sınıfları interface’ler ile tasarladık. Bir sınıfı kullanmamız gerektiğinde interfacelerini kullandık, böylece proje üzerinde daha gerçeklemeler yokken dahi sınıflarımızı tasarlayabildik ve ilerleme kaydettik. Interface’leri değiştirmemeye özen gösterdik. Bir değişiklik olacağı zaman ise birbirimize anından haber verdik. Bu sayede conflict’lerin önüne geçmiş olduk. Push’ladığımızda çalışır bir system elde ettik.

* 1. Proje Dependency İlişkisi

Proje içerisinde herbir sınıfın birbiri ile ilişkisini en aza indermek için interface ve dependency injection kullanmak bize başka sorunlar çıkardı. New keyword’ünün hangi sınıf içerisinde gerçeklenecek olması. Bu bağımlılığın da en aza indirilmesi gerekiyordu. Bunun için ise Factory desenini kullandık. Ancak factory desen bizim bu bağımlılıkları 0’a indirmemize yaramadı ama birçok yerdeki bağımlılıkları minimize edebildik. Artık bağımlılıklar sadece main.java ve computer.java sınıfları içerisinde yer alıyordu. Bu sınıflar ise bizim kod yazmamızın önüne geçmiyordu.

* 1. Zaman Modülü  
      Zaman modülü’nün olmasının sebebi process’lerin geliş zamanına göre sistemimizin çalışmasına ihtiyaç duyuyorduk. Bunu gerçekleyebilmek için asenkron şekilde çalışacak ve bize kendi yarattığımız sistemin zamanını söyleyecek bir sınıf gerekiyordu. Kendi oluşturduğumuz sistemin zamanının akışını da yine bizim kontrol etmemiz lazımdı. Bu sistemin modüler olmasının önünü açarken systemin kolay konfigüre edilebilmesine de yarıyordu. Kendi sistemimiz içerisinde her bir geçen zaman diliminde process’in gelişini, scheduling algoritmasını, waiting queue kontrolünü, cpu’nun process’i execute ( yani ekrana çıktı verme işlemi ) etmesini kontrol edecektik. Bu yüzden böyle bir zaman modülü gerçekleme ihtiyacı duyduk.
  2. Observer Deseni İle Asenkron Gerçekleme  
      Zaman geçtikçe bizi haberdar edecek bir tasarıma ihtiyacımız vardı ki burada devreye observer deseni giriyor. Publisher her bir zaman tick’i içerisinde observer’ların update() methodunu çalıştıracak. Observer’ların kendilerine ait çalışma sırası olacak ve publisher içerisinde bu sıraya göre observerlar’ı tutacak yine bu sıraya göre observerları notify() edecek. Böylece sıralamayı istediğimiz şekilde kolayca değiştirebileceğimiz modüler bir yapı olacak. Ayrıca bizim istediğimiz şekilde çalışan bir system elde edeceğiz.
  3. Bellek Algoritmamızın Seçilme Sebebi   
      Algoritmamızda tercih ettiğimiz Paging memory allocate mechanism sıklılıkla gerçek işletim sistemleri tarafından tercih edilen, gerçeklenen ve kullanılan bir mantık olduğu için bizim sistemimizde de kullanılması gerektiğine karar verdik. Peki bizce neden diğerleri değilde paging:   
     Contiguous memory allocation mekanizmasını tercih etmedik çünkü process’ler çalışmasını bitirdiğinde external fragmentation’a sebep olacaklardı ve Ram’in bir çok bölümünü kullanamayacak hale gelecektik.  
      Dynamic memory allocation mekanizmasını tercih etmememizin sebebi ise bu proje içerisinde bizlere verilen process’lerin bütün var oluş süreleri boyunca sabit miktarda bellek ve i/o device’lara ihtiyaç duyması, dinamik olarak bir verilerinin değişmemesi.

Fixed partitioning mekanizmasını tercih etmememizin sebebi ise büyük bir bellek bölgesinin sadece bir process için tahsis edilmesi ve internal fragmantation’ın ram’in bir çok kısmını kullanmamıza engel olacak olması.

Variable Partitioning seçilmemesinin sebebi dinamik bir girdimizin olmaması.

Diğer mekanizmaların seçilmemelerinin sebepleri de gerek ihtiyacımızın dışında bir sürü mekanizma içerip kurguladığımız system için ekstra yavaşlık veya karmaşıklık katacak olmaları. Bu yüzden en iyi seçim Paging mekanizması, algoritması olarak duruyordu.

* 1. Görevlendirici Yapısının Kurulması ve Olası Başka Tasarımlar  
      Görevlendirici gerçek işletim sistemlerinde olduğu gibi bizim işletim sistemimizde de CPU’yu görevlendirmek için scheduling algoritmalarından yararlanıyor. Görevlendiricimiz yani Dispatcher’ımız schedule algoritmamızdan gelen process’I CPU’ya koyma işleminden sorumlu. Asıl seçim işlemini scheduler yapıyor ve dispatcher’ı görevlendirerek context’in switch’lenmesini sağlıyor.   
      Ayrıca diğer işletim sistemlerinde olduğu gibi bizim işletim sistemimizde de modüler bir yapı mevcut. Bütün görevi tek bir modülün sorumluluğuna yüklemektense sorumlulukları OS içerisindeki ilgili yapılara dağıttık. Dispatcher ve Scheduler, bu modülleri kullanarak yapmak istedikleri işlemleri OS üzerinden çağrılar yaparak gerçekleştiriyorlar.  
      Örneğin kaynak tahsis etme işlemini dispatcher’ın sorumluluğuna vermedik – ki hiçbir örnek işletim sisteminde dispatcher’ın sorumluluğunda değil – bu görevi process manager sınıfına verdik. Kaynak tahsis etmelerden ve ilk pcb değerlerinin atanmasından bu sınıf sorumlu. Bunun gibi birçok görevi modüler bir şekilde dağıttık.
  2. Scheduler Algoritmalarının Tartışılması  
      Scheduler algoritmalarının belirli bir algoritmaya göre process’leri dispatcher’a görevlendirmek için iletmelerindeki asıl sebep process’lerin cpu üzerinde adil bir şekilde görevlendirilmesini sağlayabilmek. Burada adil olmaktan kastettiğim ready queue’ya gelmiş ancak çalışmak için cpu’da zaman ve sıra bulamayan process’lerin önüne geçmek. Bunu ise birden fazla scheduler algoritması gerçekleyerek yapıyoruz. Bizim işletim sistemimizde bu algoritmalar FCFS, Geri beslemeli kuyruk ve RoundRobin mekanizması ile sağlanıyor. Böylece herbir process ready olduğunda kısmen adil bir şekilde cpu süresi elde edebiliyor.  
      Bu scheduler algoritmalarını seçmemizin sebebi bizlere proje dokümanı içerisinde gerçeklememiz gereken algoritmaların bu olduğunun zaten söylenmiş olması. Bu algoritmalar process’lerin cpu’yu elde etme sürelerini ayarlama konusunda iyi iş çıkarıyorlar ancak gerçek zamanlı process olayı user process’lerini hiçe sayarak çalışıyor. Bu durumda user process’ler oldukça uzun zaman bekliyorlar bu kısım optimize edilebilir. Örneğin gerçek zamanlı process bir user process için 5 saniyede bir cpu’yu bırakıp yer verebilir. Bunun gibi geliştirmeler yapılabilir. Ancak bence gayet güzel iş çıkarıyor bu algoritmalar.

1. Proje Akış Mantığı  
    Publisher modülüne herbir observer kendi kendini constructor’ı içerisinde publisher’ın static attach() methodu ile ekler. Herbir observer’a ait çalışma sırasını gösteren sequence number vardır. Bu numaraya göre publisher içerisindeki listede sıralı tutulurlar. Publisher observer listesini – ki bu liste sıralı – notify eder. Bu notify işlemi herbir zaman tick’i içerisinde gerçeklenir. Zaman asenkron bir şekilde çalışmasına operating system bilgisayara yüklendiğinde başlar (instance’ı oluştuğunda). Herbir notify işlemi içerisinde sırasıyla process manager, user, scheduler, cpu, OS kendi update methodlarını çağırırlar.

Daha anlaşılabilir bir anlatım olması açısından process manager ilk tetiklenmesine ragmen son olarak onu açıklayacağım.  
 User, update()’i içerisinde verilen process’lerin geliş zamanına göre OperatingSystem’e create process çağrısı yapar. OperatingSystem bu çağrıyı kendi modüllerinden process manager içerisinde handle eder. Bu handle işlemi Ram ve I/O Device’ların kontrol edilmesiyle başlar. CheckRamandDevices() method Process’in gereksinimlerine göre process’in oluşturulabileceğini, process’in silinmesi gerektiğini veya process için o anlık yeterlik kaynak bulunmadığına ait değer döner. Yine process manager burdaki dönüş değerine göre process’i siler, bekletir – waiting queue içerisine koyması için OS’e verir – veya oluşturup gerekli kaynakları tahsis eder ve ready queue’lardan uygun olanına yerleştirmesi için OS’e iletir. OS kendisine iletilmiş bir process varsa State’inin kontrol eder. Ready ise görevi Scheduler’a devreder. Eğer Waiting ise kendisi Waiting Queue’ya yerleştirir.

SchedulerOS , update()’i içerisinde schedule algoritmalarına göre ( GBG, RR, FCFS ) queue’lardan process getirir ve dispatcher’ın context switch() methodunu çarğırarak cpu içerisinde process’i yenisiyle değiştirirken eski process sonlanmadıysa queue’lara tekrardan ekleme işlemi yapıyor.  
 CPU, update() methodu içerisinde field olarak tuttuğu currentProcess’i execute ediyor. Verilen process text’i içerisinde process’lere bir iş yapması için görev atanmadığından ekrana çıkarma işlemini process’in yaptığı iş gibi hayal ettik ve execute() methodu process’in işiymişcesine ekrana çıkarma işlemini gerçekliyor.

OperatingSystem, update() methodu içerisinde waiting queue’da bekleyen proceslerin ready queue’ya geçebilmek için ihtiyaç duydukları kaynaklar var mı diye kontrol ediyor - bunu yukarda bahsettiğim gibi modülerlik açısından da doğru olduğu için alt modüllerine yaptırıyor – varsa tahsis ediyor ve ready queue içerisine yerleştirmesi için schedule’un scheduleProcess() methoduna gönderiyor.

ProcessManager’ın update methodunu sona sakladım çünkü yaptığı iş sonlanan process’lerin kapladıkları ram bölgelerini ve kullandıkları device’ları serbest bırakmak. Bu işlemi process’lerin lifecycle’ından sorumlu olan bir modül olduğu için ProcessManager içerisinde gerçekliyoruz.

Bütün process’lerin çalışma işlemi bittiğinde, bekleyen process kalmadığında ve yeni hiçbir process gelmediğinde bilgisayarın kapanmasını gerektiğini düşündük. Bunu ise OS içerisinde queue’ları kontrol ederek gerçekliyoruz. Queue’lar boşsa ve CPU 5 kere boşa çalışmıssa bilgisayar kapanmalı. Bu durumda OS, bilgisayarı kapatma sinyali gönderiyor.

Böylece kurguladığımız küçük evren içerisindeki kullanıcımızın bilgisayarı kapanmış, işlemleri işletim sistemimiz tarafından gerçekleşmiş oluyor ve sanal kullanıcımız hayatını, biz ise hayatımızın bu kısmını noktalamış oluyoruz.

1. Uygulamaya Ait Görseller