İşletim Sistemleri Dersi

3.Sınıf
Güz Dönemi

Proje Ödev
Raporu

Hazırlayanlar
G191210035 – Zekeriya Altunkaynak
B191210046 – Necat Sungur Cihan
G181210040 – Sami Çağlar Acar
B181210033 – İzzet İlker Durdu
G191210078 – Berat Sağdıç

Proje GitHub Linki

https://github.com/SungurCihan/Grup20-OS

Aralık 2022

Sakarya Üniversitesi

Projeye giriş, içeriğin açıklanması

Eclipse IDE kullanarak, Java dilinde işletim sistemlerinin tasarım prensiplerine uygun bir biçimde bir görevlendirici tasarlamak, gerçekleme sürecinde karşılaşılan problemlere algoritmik çözümler getirmektir.

Projenin içeriğinde kısaca aşağıdaki kütüphanelerden yararlanılmıştır.

`java.util.Timer`
`java.util.TimerTask`
`java.io.BufferedReader`
`java.lang.ProcessBuilder`

Kullanılan bu sınıflar tasarımsal süreçte dispatcher'ın varan prosesleri sıraya koyma, çalıştırma ve sonlandırmasından sorumlu olmuştur.

Proje tasarımının açıklanması, projeye giriş

Projede genel anlamıyla tasarım sürecinde OOP yaklaşımı ve SOLID prensipleri göz önünde bulundurarak tasarlanmış bu sebeple modüler bir yapıya gidilmiştir.

Projede kullanılan katmanların genel tanımı aşağıdaki gibidir.

Core.File : Jar dosyasına komut satırı parametresi olarak verilen txt dosyasının okunup ayrıştırılarak belleğe alınması, alınırken her process için PID ataması gibi işlemleri karşılayan katmandır.

Core.Utilities : Yardımcı ve genel fonksiyonların bulunduğu katmandır.

Core.Queue : Proses kuyruklarının tutulduğu ve kuyruk havuzundaki algoritmaların çalıştırıldığı katmandır.

Core.Dispatcher: Bu katmanda timer sınıfı kullanılarak dosyadan okunan proseslerin varma durumlarına ve önceliklerine göre sıraya alınarak değerlendirilme işlemleri gerçekleştirilmiştir. İlk ifadeyle prosesler okunduktan sonra burda işlem görmektedirler.

Core.Process : **En kilit katmanın bu olduğu söylenebilir**. Proseslere dair özelliklerin, tanımların ve çalışacak fonksiyonların ana hedefi bu katmandır.

Core.Main : İşletim sisteminin görevlendiriciyi ilk aşamada çalıştırabilmesi için gerekli olan katmandır.

Projenin Tanıtılması, kodların değerlendirilmesi

Programın Çalıştırılması

```
1  package Core.Main;
2
3  import java.io.IOException;
8  public class Program {
9
10     public static void main(String[] args) throws IOException {
11         //Veriler dosyadan okunuyor
12         ReadFromFile.ReadFile(args[0].toString());
13
14         //Zamanlayıcı ve dispatcher başlatılıyor.
15         Dispatcher a = new Dispatcher();
16         a.StartTimer();
17     }
18
19 }
20
```

Görevlendirici dosyadan process bilgilerini okuduktan sonra zamanlayıcı başlatarak işlemleri değerlendiriyor.

Buradaki önemli nokta, okutulacak olan veri dosyası CLI üzerinden parametre olarak alınmaktadır. Çalıştırılan jar dosyasıyla aynı konumda bulunan bir dosya parametre olarak verildiğinde dosya okutulacaktır.

NOT: Renkli arayüzün Windows üzerinde çalıştırılabilmesi için VirtualCommand parametresi kayıt defterine eklenmiş olmalı , aynı zamanda cmd.exe ile değil conhost.exe ile run edilmelidir.

Dosya Okuma Sistemi

```
package Core.File;

import java.io.BufferedReader;
public final class ReadFromFile {

    //Dosyadan veri okunan fonksiyon
    public static void ReadFile(String filePath) throws IOException
    {
        QueuePool.InitializeQueue();
        File file = new File(filePath);
        FileReader fileReader = new FileReader(file);
        String line;
        int number=-1;
        BufferedReader br = new BufferedReader(fileReader);
        while ((line = br.readLine()) != null)
        {
            line = line.replaceAll("\\s", "");
            number++;
            String[] parsedProcess = line.split(",");
            Process process = new Process();
            process.ArrivalTime = Integer.valueOf(parsedProcess[0]);
            process.Priority = Helpers.GetPriorityEnum(Integer.valueOf(parsedProcess[1]));
            process.BurstTime = Integer.valueOf(parsedProcess[2]);
            process.ProcessId = Helpers.CreatePid(number);
            QueuePool.add(process);
        }
        br.close();

        //Dosyadan okunup kuyuklara eklenen prosesler varış zamanlarına göre sıralanıyor.
        QueuePool.SortAllQueues();
    }

}
```

Kodlardan anlaşılacağı üzere, path olarak komut satırından girilen text dosyası satır satır okunuyor, kırmızı ile belirttiğim yerde, virgülle ayırma aşamasında hocamız birer boşluk karakteri bıraktığı için olası hatalara mahal vermemek amacıyla ek bir kontrol gerçekleştirilmiştir. Process olarak oluşturduğumuz sınıftan her satır için bir instance üreterek Kuyruğa ekleniyor.

Bu aşamada **QueuePool** olarak çağırılan final bir class mevcuttur, bu classın içindeki tanımlar statik olup kodlama aşamasında OOP tasarımına bağlı kalmak amaçlanmıştır.

Aynı sınıfın içinde SortAllQueues() adında bir fonksiyon mevcuttur, fonksiyonun kodları aşağıda mevcuttur.

```
104     //Kuyukların içersindeki prosesler varış zamanlarına göre sıralandığı fonksiyon
105     public static void SortAllQueues() {
106         Helpers.bubbleSortWithArrivalTime(RealTimeQueue);
107         Helpers.bubbleSortWithArrivalTime(UserBasedQueue1);
108         Helpers.bubbleSortWithArrivalTime(UserBasedQueue2);
109         Helpers.bubbleSortWithArrivalTime(UserBasedQueue3);
110
111     }
```

Genel deyimiyle Utilities yani yardımcı fonksiyon ve bileşenlerin yazıldığı katmanda Helpers adında bir class yapısı mevcuttur, bu yapı sayesinde

dosyadan rastgele okunan veriler veri yapıları dersinden öğrendiğimiz BubbleSort algoritmasıyla kuyruklar Varış zamanına göre sıralanmaktadır.

Bu sayede görevlendiricinin üzerindeki yük henüz başlama aşamasında bir bakıma azaltılmıştır. Bu bir optimizasyon örneğidir.

Burada karışıklığa yol açmamak için QueuePool.add() fonksiyonu içinde okunan her prosesin öncelik verisine göre ayrıştırılarak 4 ayrı kuyruğa dağıtıldığını belirtmek istiyoruz.

```
    //Parametre olarak alının öncelik değerine göre ilgili kuyruğa
    //yine parametre olarak aldığı "Proses" verisini ekleyen fonksiyon.
    public static void add(Core.Process.Process process) {
        switch(process.Priority)
        {
        case P0:
            RealTimeQueue.add(process);
            break;
        case P1:
            UserBasedQueue1.add(process);
            break;
        case P2:
            UserBasedQueue2.add(process);
            break;
        case P3:
            UserBasedQueue3.add(process);
            break;
        }
    }
```

Bu noktaya geldiğimizde artık elimizde dosyadan okunarak 4 ayrı kuyruğa ayrıştırılmış ve varış zamanına göre sıralanmış prosesler mevcuttur.

Bu noktadan itibaren Dispatcher'a geçebiliriz.

Program.javadan başlatılan timer Dispatcher'ı tetikler ve sistem işlemeye başlar. (Timer her 1 saniyede bir dönmektedir. Çünkü min zaman 1 Quantum zamandır. Anlaşılması açısından gerçek hayattaki 1 saniye baz alınmıştır.)

```java
public class Dispatcher extends TimerTask {
    //Zamanlayıcı değişkenleri
    private static Timer Timer;
    private int timerCounter=0;
    @Override
    public void run() {
        System.out.println("Saniye  = "+timerCounter);
        CheckInterrupted(); //Zaman aşımına uğrayan proses olup olmadığı kontrol ediliyor.
        ClearTerminated(); //Zaman aşımına uğradığı için "TERMINATED" durumuna çekilmiş lakin kuyruktan sillinmemiş proses va
        LoopingQueues(); //Bütün kuyruklar dönülerek o saniye içerisinde varan proseslerin durumu "READY" olarak değiştiril
        timerCounter++;

        //İçinde bulunduğumuz saniye içerinde çalışmaya hazır durumda olan,
        //dolayısıyla "Running Process" değişkeni içerisine yerleşmiş olan proses yürütülüyor.
        if(QueuePool.RunningProcess!=null)
        {
            try {
                QueuePool.RunningProcess.Run(timerCounter);
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

        //Yürütülecek herhangi bir proses kalmadıysa görevlendirici sonlandırılıyor.
        if(QueuePool.CheckIfSchedulerEnded())
            {
            System.out.println("Gorevlendirici vazifesini laigiyla ifa etmek hususunda kiymet-i harbiyesi ala bir muvaffakiyet
            Timer.cancel();
            }
    }
```

Burada ilk nokta timer değerini her arttırdığında her saniye ready durumuna gelen prosesslerin kontrolü yapılıyor, zaman aşımı kontrolü yapılıyor ve o an çalışan bir proses olup olmadığı kontrol ediliyor.

Şartlı ifadede görevlendiricinin içine konan bir proses olup olmadığı kontrol ediliyor**. Bu durum da bir optimizasyon örneğidir**, RunningProcess adında process tipinde tanımlanmış statik bir değişken sayesinde kontrol kolaylaştırılmış ve karmaşık döngülerden kaçınılmıştır.

Resimde işaretlendiği sırada değerlendirmeye devam edecek olursak , Timer sınıfına sahip run fonksiyonunun kontrol ettiği son bileşen , yürütülecek herhangi bir proses olup olmadığını kontrol eden fonksiyon gösterilebilir.

```
private void LoopingQueues() {

    if(QueuePool.getRealTimeQueue().getSize() != 0) //Kuyruk boş ise içeri girilmeyecek
    {
        for(int i = QueuePool.getRealTimeQueue().first;i <= QueuePool.getRealTimeQueue().last;i++)
        //Kuyruk dönülüyor
        {
            Process currentProcess=QueuePool.getRealTimeQueue().getProcess(i); //Şu an üzerinde bulunulan proses
            Process nextProcess=QueuePool.getRealTimeQueue().getProcess(i+1);  //Bir sonraki proses

            //Proses bu saniye içerinde varmışsa durumu "READY" yapılıyor
            if(currentProcess.ArrivalTime == timerCounter)
            {
                currentProcess.ChangeState(State.READY);

                //Çalışmakta olan başka bir proses yoksa veya daha düşük öncelikli bir proses
                //çalışmaktaysa şu an varmış olan proses "RunningProcess" değişkenine atanarak
                //çalışmaya hazır olduğu ilan edilir.
                if(QueuePool.RunningProcess==null||QueuePool.RunningProcess.Priority!=Priority.P0)
                    QueuePool.RunningProcess=currentProcess;

                //Bir sonraki proses, içerisinde bulunduğumuz saniye varmıyorsa döngü kırılır
                //ve kuyuruğun geri kalanı boşuna dönülmez. Bu iddia da bulanilmemizin sebebi
                //kuyruktaki prosesleri daha önce varış zamanlarına göre sıralamış olmamız.
                if(nextProcess!=null&&nextProcess.ArrivalTime!=timerCounter)
                    break;
            }
        }
    }
```

OPTİMİZASYON ÖRNEĞİDİR.

Her saniye, varan bir proses varsa durumunu READY olarak değiştiren **LoopingQueues** fonksiyonumuzda değerli bir optimizasyon mevcuttur. Bu aşamada proseslerin varış zamanı kontrol edilirken, kuyruk yapısının bize sağladığı kolaylık kullanılarak, daha önce bubble sort algoritmasıyla varış zamanına göre sıralanmış prosesler kontrol edilmekte, bir sonrakinin varış zamanı şuandan farklı ise bütün kuyruğu dolanmak yerine döngü kırılmış, bu sayede kayda değer bir verim elde edilmiştir. Kodun devamında diğer kuyruklar kontrol edilmiştir.

```
private void CheckInterrupted() {
    for (ProceessInterruptedModel item : QueuePool.getInterruptedProcesses()) {
    //Askıda olan proseslerin bulundğu dizi dönülüyor
        if(timerCounter - item.InterruptedTime == 20) //Zaman aşımına uğrama durumu tespit ediliypr
        {
            switch(item.Priority)
            //Zaman aşımına uğrayan proses öncelik durumuna göre ilgili kuyuruktan siliniyor.

            //Bir proses askıya alınmışsa, bu onun daha kullanıcı menşeli bir proses olduğuna
            //ve daha önce çalışmış olduğuna delalet eder. Dolayısıyla bu proses ya "2" ya da "3"
            //önceliğindedir. Bu iki ihtimal ayrı birer durum telakki edilerek ele alınıp
            //sonlandırılma işlemi gerçekleştiriliyor.
            {
            case P2: //Prosesin "2" önceliğinde olduğu durum
            {
                for(int i = 0; i < QueuePool.getUserBasedQueue2().getSize(); i++)
                //"2" önceliğinde olan proseslerin tutulduğu kuyruk dönülüyor.
                {
                    if(item.ProcessId == QueuePool.getUserBasedQueue2().getProcess(i).ProcessId)
                    //İlgili proses tespit edilince durumu "TERMINATED" olarak değşitirilip
                    //ekrana gerekli mesaj basılıyor.

                    //Kuyuk veri yapısı işleyiş mantığı olarak kuyruğa eklenen ilk elemanı siler.
                    //Burada zaman aşımına uğramış olan prosesin kuyruğun ilk elemanı olduğunu vaat
                    //edemediğimizden kuyruktan silme işlemi burada gerçekleştirilmiyor. Bu proses
                    //kuyruğun ilk elemanı olduğu zaman bu durum tespit edilerek proses kuryuktan çıkarılacak.
                    //220. satırdaki "ClearTerminated" fonksiyonu bu amaçla yazıldı.
                    {
                        Process newProcess = QueuePool.getUserBasedQueue2().getProcess(i);
                        newProcess.State = State.TERMINATED;
                        QueuePool.getUserBasedQueue2().change(i, newProcess);
                        newProcess.PrintTimeout();
                    }
                }
            }
```

CheckInterrupted fonksiyonunda ödev dosyasında belirtildiği gibi, varıp çalıştıktan sonra 20 saniye boyunca çalışmayan fonksiyonların kontrolü sağlanmaktadır. Şartlı ifadeden anlaşılacağı üzere şuanki zamandan prosesin son çalıştığı zamanın farkı alınarak 20'ye eşit olma durumunda proses INTERRUPTED durumundan TERMINATED durumuna çekiliyor.

Kuyruk yapısına aykırı hareket etmemek ve aradan çıkarma işlemleriyle gereksiz döngülerden kaçınmak amacıyla, ClearTerminated() fonksiyonu devreye girmektedir.

Bu fonksiyon, UserBased2 ve UserBased3 kuyruklarının ilk elemanlarını kontrol ederek sadece terminated durumunda olan prosesler kaldıysa, onları kuyruktan çıkarıyor.

Bu noktada UserBased1 kuyruğunun kontrol edilmeme sebebi, birinci önceliğe sahip bir prosesin en az bir kez çalıştıktan sonra önceliği düşürülüp 20 saniyelik süreci başlayacağından ötürü, sadece 2 ve 3 önceliğine sahip kuyruklar kontrol edilmektedir. Fonksiyonun içeriği aşağıdadır.

```
private void ClearTerminated() {
    //Kuyrupun ilk elemanının durumunun "TERMINATED" olması halinde
    //kuyruktan silinme işlemi gerçekleştiriliyor. Bunun sebebi yukarıda(186. satır)
    //detaylı olarak açıklandı

    if(QueuePool.getUserBasedQueue2().getProcess(QueuePool.getUserBasedQueue2().first) != null
            && QueuePool.getUserBasedQueue2().getProcess(QueuePool.getUserBasedQueue2().first).State == State.TERMINATED)
        QueuePool.getUserBasedQueue2().remove();

    if(QueuePool.getUserBasedQueue3().getProcess(QueuePool.getUserBasedQueue3().first) != null &
            & QueuePool.getUserBasedQueue3().getProcess(QueuePool.getUserBasedQueue3().first).State == State.TERMINATED)
        QueuePool.getUserBasedQueue3().remove();
}
```

Process Sınıfı'nın Açıklanması

Kuyruk kısmına geçmeden önce process sınıfının özelliklerine ve içerdiği fonksiyonlara değinmek daha verimli olacaktır.

```
//Proses sınıfı tanımlanıyor
public class Process {
    public String ProcessId;                //PID
    public int ArrivalTime;                 //Varış Zamanı
    public int BurstTime;                   //İcra edilceği toplam süre
    public Priority Priority;               //Öncelik
    public State State;                     //Durum
    private ProcessBuilder processBuilder;  //İşletim sistemi proses başlatmak için gerekli sınıf

    //Kurucu sınıf içerisinde "ProcessBuilder" değişkenin içerisi doldurulуyor.
    public Process() {
        this.processBuilder= new ProcessBuilder();
    }
```

Proseslerin özellikleri **public property** olarak tanımlanarak, gerek dosya okuma aşamasında, gerek dispatcher yürütülürken içeriğine erişilmiş ve prosese ait olabilecek durumların kontrolüne olanak sağlanmıştır.

Örneğin ; bir prosesin çalışması, durumunun değişmesi, ekrana yazılması, timeout olması, kesilmesi veya çalışan olarak atanması gibi örnekler verilebilir.

Ödev dökümanında proseslerin Java Proses'i olarak tanımlanması istendiğinden, her proses nesnesi oluştuğunda, özelliklerine ek olarak bir ProcessBuilder nesnesi beraberinde oluşturulmuştur. Biz projemizde bu java prosesine ekrana yazma görevi yükledik.

```
public void Run(int currentTime) throws IOException {
    switch(this.Priority)
    {
    case P0: //REALTIME
    {
        //Durum "RUNNING" olarak değiştirilip çalıştırılıyor.
        ChangeState(Core.Utilities.State.RUNNING);
        StartProcess();
        break;
    }
    case P1: //USER - BASED (PRIORITY 1 )
    {
        this.State=Core.Utilities.State.RUNNING;
        StartProcess();
        if(this.BurstTime!=0)
        {
            //Eğer proses noktalanmadıysa 1 saniye çalıştırıldıktan sonra
            //askıya alınıyor ve önceliği düşürülüp, içinde bulunduğu kuyruktan
            //silinmek suretiyle bir alt kuyruğa ekleniyor.
            QueuePool.remove(this.Priority);
            this.Priority=Core.Utilities.Priority.P2;
            QueuePool.add(this);
            ChangeState(Core.Utilities.State.INTERRUPTED);
            QueuePool.getInterruptedProcesses().add(new ProceessInterruptedModel(this.ProcessId, currentTime, Core.Utilit
        }
```

Proses özelliği taşıyan her nesne Run özelliği içerir. Burada prosesin priority verisine göre durumu running'e getirilmiştir. **Ödevin kaynak kodları içerisinde bu durumlar ayrıntılı olarak açıklandığından burada anlatılmamıştır.**

```
//Durumu değişen fonksiyonun yeni durumunun ve diğer bilgilerinin ekrana basıldığı fonksiyon
private void Print()
{
    String processInformation="(id:"+this.ProcessId+"    oncelik: "+this.Priority.toString()+"  kalan sure: "+this.BurstTi
    switch(this.State)
    {
    case INTERRUPTED:
    {
        PrintColored(Constants.MESSAGE_INTERRUPTED+processInformation);
        break;
    }
    case READY:
    {
        PrintColored(Constants.MESSAGE_READY+processInformation);
        break;
    }
    case TERMINATED:
    {
        PrintColored(Constants.MESSAGE_TERMINATED+processInformation);
        break;
```

Print() fonksiyonunda ödev dökümanındaki formata uygun şekilde proses bilgisi konsola yazdırılmıştır. Burada her nesne eşsiz bir PID'ye sahip olduğundan bu durumdan yararlanılarak renk şemaları da her proses için ayrı oluşturulabilmiştir, Örnek çıktı ödev dökümanının sonunda mevcuttur.

```
//Prosesin yürütüldüğü fonksiyon
private void StartProcess() throws IOException
{
    //Prosesin an itibariyle yürütüldüğü bilgisini ekrana basan gerçek bir işletim
    //sistemi prosesi başlatılıyor.
    String processInformation="(id:"+this.ProcessId+"    oncelik: "+this.Priority.toString()+"  kalan sure: "+this.BurstTi
    String message = Constants.MESSAGE_RUNNING+processInformation;
    processBuilder.command("cmd", "/c", "echo "+ message);
    java.lang.Process process = processBuilder.start();
    ReadProcess(process);

    //Prosesin sonlandırılmış olma ihtimali değerlendiriliyor.
    this.BurstTime--;
    if(this.BurstTime==0)
    {
        ChangeState(Core.Utilities.State.TERMINATED);
        QueuePool.remove(this.Priority);
        SetRunningProcess();
        return;
    }
```

Dispatcher tarafından çalıştırılan proses, fonksiyona düştüğünde prosesin bilgisi Java Proses sınıfından faydalanarak ekrana yazılmıştır. Ardınan BurstTime'ı azaltılarak prosesin bitip bitmediği kontrol edilmiştir.

Process sınıfına dair gösterilecek bir diğer önemli fonksiyon SetRunningProcess() olacaktır.

Bu fonksiyon her saniye hangi prosesin çalışacağına karar veren önemli bir fonksiyondur, bu aşamada da optimizasyon mevcuttur.

```
private void SetRunningProcess() {

    //Her kuyruğun ilk elemanı getiriliyor.
    Process firstItemOfRealTimeQueue=QueuePool.GetFirstItem(Core.Utilities.Priority.P0);
    Process firstItemOfUserBased1Queue=QueuePool.GetFirstItem(Core.Utilities.Priority.P1);
    Process firstItemOfUserBased2Queue=QueuePool.GetFirstItem(Core.Utilities.Priority.P2);
    Process firstItemOfUserBased3Queue=QueuePool.GetFirstItem(Core.Utilities.Priority.P3);


    //İlk eleman boş değilse ve çalışmaya hazır durumda ise gerçek zamanlı olana öncelik
    //vererek çalışma imkanı sunuluyor.
    if(firstItemOfRealTimeQueue!=null&&firstItemOfRealTimeQueue.State==Core.Utilities.State.READY)
        QueuePool.RunningProcess=firstItemOfRealTimeQueue;

    else if(firstItemOfUserBased1Queue!=null&&firstItemOfUserBased1Queue.State==Core.Utilities.State.READY)
        QueuePool.RunningProcess=firstItemOfUserBased1Queue;

    else if(firstItemOfUserBased2Queue!=null&&
            (firstItemOfUserBased2Queue.State==Core.Utilities.State.READY||
            firstItemOfUserBased2Queue.State==Core.Utilities.State.INTERRUPTED))
        QueuePool.RunningProcess=firstItemOfUserBased2Queue;

    else if(firstItemOfUserBased3Queue!=null&&
            (firstItemOfUserBased3Queue.State==Core.Utilities.State.READY||
            firstItemOfUserBased3Queue.State==Core.Utilities.State.INTERRUPTED))
        QueuePool.RunningProcess=firstItemOfUserBased3Queue;

    else
        QueuePool.RunningProcess=null;
```

Bu aşamada ödev dökümnanından yola çıkarak önceliklere göre algoritmik ifadeler yazılmıştır. Her saniye önce real time kuyruk kontrol edilmektedir.

Kuyruk Havuzu, Kuyruk Sınıfı'nın açıklanması

Kuyruk veri yapısı Core.Queue katmanında tarafımızdan gerçeklenmiş, ve özelliklerine uygun biçimde kullanılmıştır.

```
//Proseslerin içerisinde tutulacağı "Kuyruk Sınıfı" tanımlanıyor.
public class Queue {
    public int first; //Kuyruğun ilk elemanı
    public int last;  //Kuyruğun son elemanı
    int capacity;     //Kuyruğun kapasitesi(içerebileceği maksimum eleman sayısı)
    Process array[];  //Kuyruğun içerisinde tutlacak olan veri(Prosesler)
    int processCount; //Kuyrupun içerisinde bulunan proses sayısı

    //Yapıcı fonksiyon içerisinde varsayılan atamalar yapılıyor.
    public Queue(){
        this.first = 0;
        this.last=-1;
        this.array = new Process[1];
        this.capacity = 0;
        this.processCount=0;
        expand(5);
    }

    //İlgili indexdeki prosesi getiren fonksiyon
    public Process getProcess(int index)
    {
        if(array.length>index)
            return array[index];
        return null;
    }
```

Bütün sınıfı açıklamak ve anlatmak yerine bu noktada özet geçmek gerekirse;

Kuyruk sınıfı dinamik olarak tanımlanmıştır, ve ekleme aşamasında expand() fonksiyonu kullanılarak genişlemeye ihtiyaç varsa otomatik olarak kuyruk genişletilmiştir.

Yine kuyruk yapısına uygun add ve remove fonksiyonları mevcuttur, add sona eklerken remove kuyruğun başından veri silmektedir(indexi kaydırmaktadır).

```
//Uyglamanın hayat döngüsü boyunca kullanılan kuyruk ve dizilerin tutulduğu static sınıf
public final class QueuePool {
    //Proseslerin tutulduğu kuyuruklar
    private static Queue RealTimeQueue;
    private static Queue UserBasedQueue1;
    private static Queue UserBasedQueue2;
    private static Queue UserBasedQueue3;
```

Uygulama hayat döngüsü boyunca, belleği gereksiz yere işgal etmemek ve kod karmaşasını engellemek amacıyla private olarak Kuyruk yapıları statik bir biçimde tanımlanmıştır. Her biri için getter ve setter yazılmıştır.

(Soyutlamak için)

Ödevin bize kattığı en önemli bilginin bu olduğunu söyleyebiliriz. Daha önce C# dersinde bir programın çalıştırılabilir olması için bazı statik değişkenlere ihtiyacı olduğu öğrenmiştik ancak sebebini bilmiyorduk, bu sayede işletim sisteminin gerçek tasarımında da araştırmalarımız sonucu çok fazla statik değişkene ihtiyaç duyulduğunu öğrenmiş olduk.

Bu sınıfın fonksiyonları, add ve remove fonksiyonlarından oluşmaktadır. Kuyruklara direk veri eklemenin önüne geçilmiş ve elimizden geldiği kadar, fonksiyonları ve sınıfları soyutlama yoluna gittik, gerçek tasarımları incelediğimizde durumun böyle olması gerektiğine kanaat getirdik.

Netice itibariyle, private olarak oluşturulan kuyrukların özelliklerine sınıf içerisinden ulaşılması sağlanmıştır.

Bitirirken Bazı Notlar ve Açıklamalar

Bu projeyi gerçeklerken dikkat ettiğimiz en önemli nokta kod tekrarından kaçınmak, algoritmik ifadeleri yazarken kodun modülaritesini düşürmemekti.

Bu sebeple birden fazla kullanılan her satır kodu Core.Utilities paketi altında topladık ve buradan erişim sağladık.

```
//Proje de kullanılan sabit değişkenlerin tutulduğu static sınıf
public final class Constants {
    //Konsola renlki çıktı almak için tanımlanan değişkenler
    public static final String COLOR_SPACE_PREFIX="\u001B[38;5;";
    public static final String COLOR_SPACE_SUFFIX="\u001b[0m";

    //Prosesin durum değişimlerinde ekrana basılacak olan ifadeler.
    public static final String MESSAGE_INTERRUPTED="proses askiya alindi.";
    public static final String MESSAGE_RUNNING="proses yurutuldu. ";
    public static final String MESSAGE_READY="proses hazir durumuna gecti. ";
    public static final String MESSAGE_TERMINATED="proses sonlandirildi. ";
    public static final String MESSAGE_TIMEOUT="proses zaman asimina ugradigi icin sonlandirildi. ";
}
```

Örneğin string ifadelerde dahi, böyle bir yol izlenmesi kodu verimli ve okunabilir kılmıştır.

Bununla birlikte enum tipi kullanılarak kod okunurluğu arttırılmıştır.

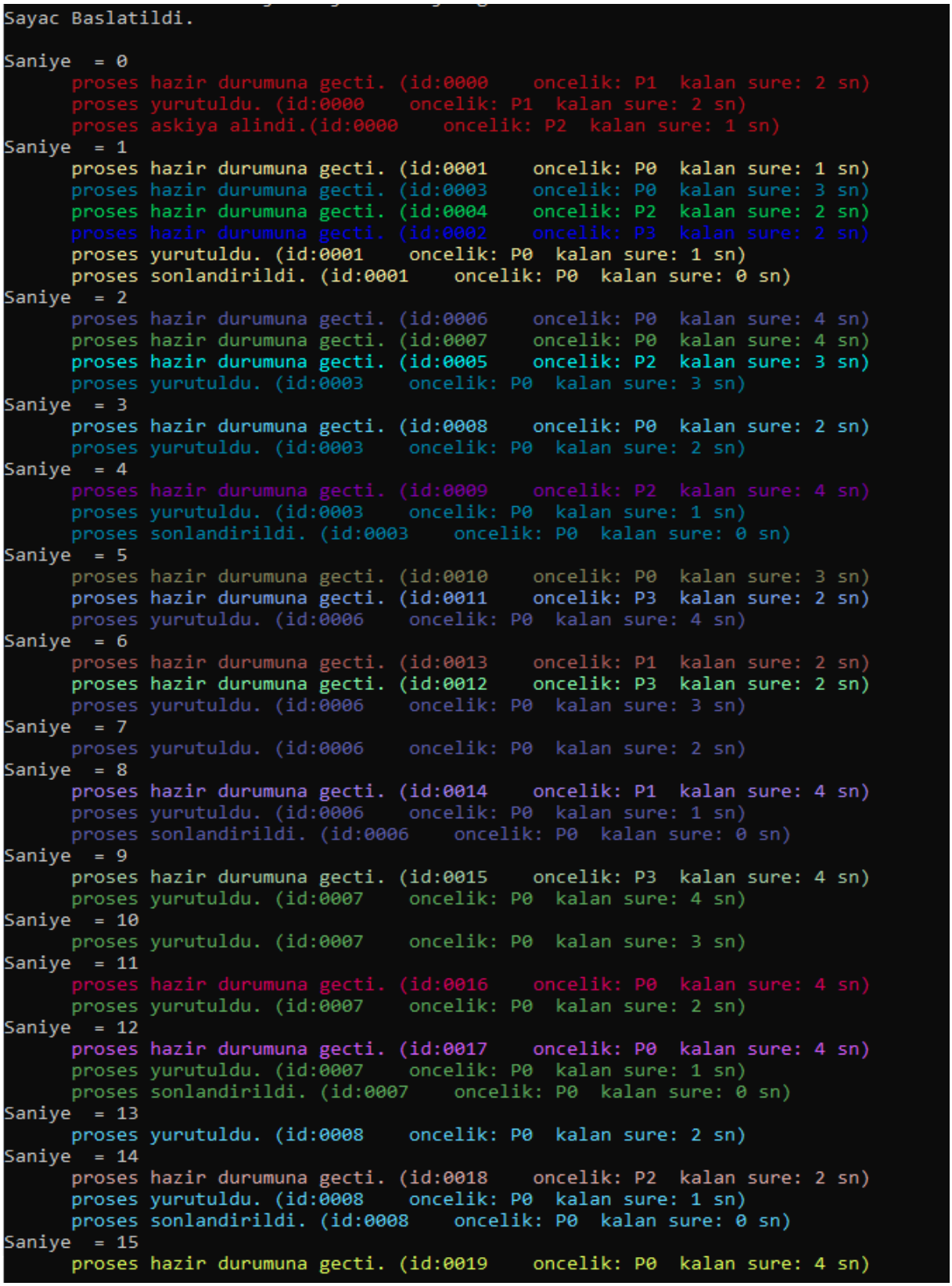
```
public final class Helpers {

    //Proses id 4 basamaklı hale getiriliyor.
    public static String CreatePid(int pid)
    {
        String temp = String.valueOf(pid);

        int length =4-temp.length();
        for(int i =0;i<length;i++) {
            temp="0"+temp;

        }
        return temp;
    }

    //Girilen integer degeri için ilgili Priority değerini döndürüyor.
    public static Priority GetPriorityEnum(int number)
    {
        switch(number)
        {
        case 0:
            return Priority.P0;
        case 1:
            return Priority.P1;
        case 2:
            return Priority.P2;
        case 3:
            return Priority.P3;
        default:
            return null;
        }
    }
```

Helpers sınıfında da aynı şekilde, kodun kalitesini artırıcı yönde bazı fonksiyonlar yazılmış ve yaşam döngüsü boyunca kullanılmıştır.

Ödev bizlerde yeni ufuklar açmış ve bakış açımızı genişletmiştir, ödevin fikir sahibi olan hocalarımıza çok teşekkür ederiz ☺

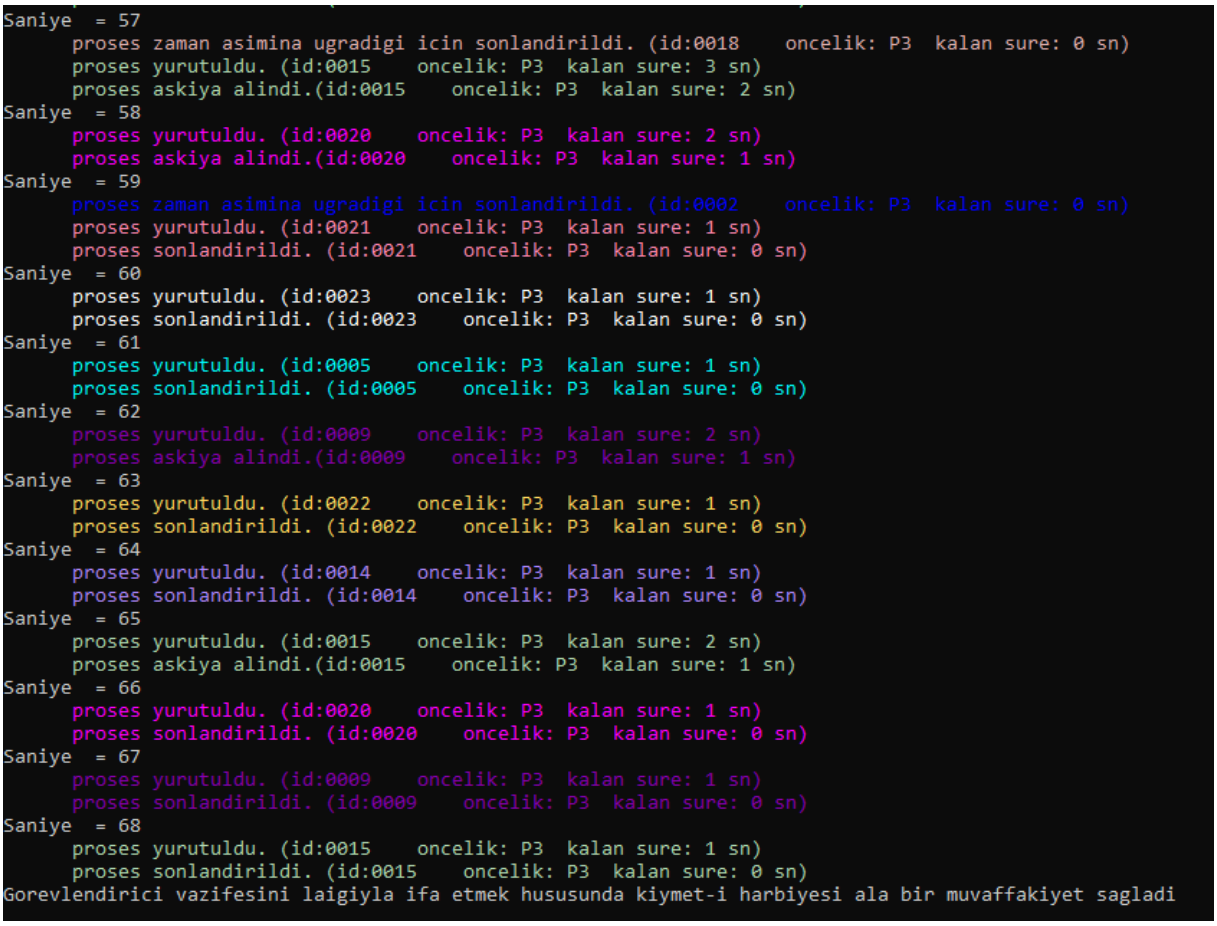
Örnek Ekran Çıktısı

Ödev dokümanında verilen örnek veri kümesinin projemizdeki çıktısı aşağıda mevcuttur.



```
Saniye  = 15
      proses hazir durumuna gecti. (id:0019    oncelik: P0  kalan sure: 4 sn)
      proses yurutuldu. (id:0010    oncelik: P0  kalan sure: 3 sn)
Saniye  = 16
      proses hazir durumuna gecti. (id:0020    oncelik: P3  kalan sure: 3 sn)
      proses yurutuldu. (id:0010    oncelik: P0  kalan sure: 2 sn)
Saniye  = 17
      proses yurutuldu. (id:0010    oncelik: P0  kalan sure: 1 sn)
      proses sonlandirildi. (id:0010    oncelik: P0  kalan sure: 0 sn)
Saniye  = 18
      proses hazir durumuna gecti. (id:0021    oncelik: P3  kalan sure: 2 sn)
      proses yurutuldu. (id:0016    oncelik: P0  kalan sure: 4 sn)
Saniye  = 19
      proses yurutuldu. (id:0016    oncelik: P0  kalan sure: 3 sn)
Saniye  = 20
      proses yurutuldu. (id:0016    oncelik: P0  kalan sure: 2 sn)
Saniye  = 21
      proses zaman asimina ugradigi icin sonlandirildi. (id:0000    oncelik: P2  kalan sure: 1 sn)
      proses yurutuldu. (id:0016    oncelik: P0  kalan sure: 1 sn)
      proses sonlandirildi. (id:0016    oncelik: P0  kalan sure: 0 sn)
Saniye  = 22
      proses hazir durumuna gecti. (id:0022    oncelik: P2  kalan sure: 3 sn)
      proses yurutuldu. (id:0017    oncelik: P0  kalan sure: 4 sn)
Saniye  = 23
      proses hazir durumuna gecti. (id:0023    oncelik: P3  kalan sure: 2 sn)
      proses yurutuldu. (id:0017    oncelik: P0  kalan sure: 3 sn)
Saniye  = 24
      proses hazir durumuna gecti. (id:0024    oncelik: P1  kalan sure: 2 sn)
      proses yurutuldu. (id:0017    oncelik: P0  kalan sure: 2 sn)
Saniye  = 25
      proses yurutuldu. (id:0017    oncelik: P0  kalan sure: 1 sn)
      proses sonlandirildi. (id:0017    oncelik: P0  kalan sure: 0 sn)
Saniye  = 26
      proses yurutuldu. (id:0019    oncelik: P0  kalan sure: 4 sn)
Saniye  = 27
      proses yurutuldu. (id:0019    oncelik: P0  kalan sure: 3 sn)
Saniye  = 28
      proses yurutuldu. (id:0019    oncelik: P0  kalan sure: 2 sn)
Saniye  = 29
      proses yurutuldu. (id:0019    oncelik: P0  kalan sure: 1 sn)
      proses sonlandirildi. (id:0019    oncelik: P0  kalan sure: 0 sn)
Saniye  = 30
      proses yurutuldu. (id:0013    oncelik: P1  kalan sure: 2 sn)
      proses askiya alindi.(id:0013    oncelik: P2  kalan sure: 1 sn)
Saniye  = 31
      proses yurutuldu. (id:0014    oncelik: P1  kalan sure: 4 sn)
      proses askiya alindi.(id:0014    oncelik: P2  kalan sure: 3 sn)
Saniye  = 32
      proses yurutuldu. (id:0024    oncelik: P1  kalan sure: 2 sn)
      proses askiya alindi.(id:0024    oncelik: P2  kalan sure: 1 sn)
Saniye  = 33
      proses yurutuldu. (id:0004    oncelik: P2  kalan sure: 2 sn)
      proses askiya alindi.(id:0004    oncelik: P3  kalan sure: 1 sn)
Saniye  = 34
      proses yurutuldu. (id:0005    oncelik: P2  kalan sure: 3 sn)
      proses askiya alindi.(id:0005    oncelik: P3  kalan sure: 2 sn)
Saniye  = 35
      proses yurutuldu. (id:0009    oncelik: P2  kalan sure: 4 sn)
      proses askiya alindi.(id:0009    oncelik: P3  kalan sure: 3 sn)
Saniye  = 36
      proses yurutuldu. (id:0018    oncelik: P2  kalan sure: 2 sn)
      proses askiya alindi.(id:0018    oncelik: P3  kalan sure: 1 sn)
```

```
Saniye  = 37
      proses yurutuldu. (id:0022    oncelik: P2  kalan sure: 3 sn)
      proses askiya alindi.(id:0022    oncelik: P3  kalan sure: 2 sn)
Saniye  = 38
      proses yurutuldu. (id:0002    oncelik: P3  kalan sure: 2 sn)
      proses askiya alindi.(id:0002    oncelik: P3  kalan sure: 1 sn)
Saniye  = 39
      proses yurutuldu. (id:0013    oncelik: P2  kalan sure: 1 sn)
      proses sonlandirildi. (id:0013    oncelik: P2  kalan sure: 0 sn)
Saniye  = 40
      proses yurutuldu. (id:0014    oncelik: P2  kalan sure: 3 sn)
      proses askiya alindi.(id:0014    oncelik: P3  kalan sure: 2 sn)
Saniye  = 41
      proses yurutuldu. (id:0024    oncelik: P2  kalan sure: 1 sn)
      proses sonlandirildi. (id:0024    oncelik: P2  kalan sure: 0 sn)
Saniye  = 42
      proses yurutuldu. (id:0011    oncelik: P3  kalan sure: 2 sn)
      proses askiya alindi.(id:0011    oncelik: P3  kalan sure: 1 sn)
Saniye  = 43
      proses yurutuldu. (id:0012    oncelik: P3  kalan sure: 2 sn)
      proses askiya alindi.(id:0012    oncelik: P3  kalan sure: 1 sn)
Saniye  = 44
      proses yurutuldu. (id:0015    oncelik: P3  kalan sure: 4 sn)
      proses askiya alindi.(id:0015    oncelik: P3  kalan sure: 3 sn)
Saniye  = 45
      proses yurutuldu. (id:0020    oncelik: P3  kalan sure: 3 sn)
      proses askiya alindi.(id:0020    oncelik: P3  kalan sure: 2 sn)
Saniye  = 46
      proses yurutuldu. (id:0021    oncelik: P3  kalan sure: 2 sn)
      proses askiya alindi.(id:0021    oncelik: P3  kalan sure: 1 sn)
Saniye  = 47
      proses yurutuldu. (id:0023    oncelik: P3  kalan sure: 2 sn)
      proses askiya alindi.(id:0023    oncelik: P3  kalan sure: 1 sn)
Saniye  = 48
      proses yurutuldu. (id:0004    oncelik: P3  kalan sure: 1 sn)
      proses sonlandirildi. (id:0004    oncelik: P3  kalan sure: 0 sn)
Saniye  = 49
      proses yurutuldu. (id:0005    oncelik: P3  kalan sure: 2 sn)
      proses askiya alindi.(id:0005    oncelik: P3  kalan sure: 1 sn)
Saniye  = 50
      proses yurutuldu. (id:0009    oncelik: P3  kalan sure: 3 sn)
      proses askiya alindi.(id:0009    oncelik: P3  kalan sure: 2 sn)
Saniye  = 51
      proses yurutuldu. (id:0018    oncelik: P3  kalan sure: 1 sn)
      proses sonlandirildi. (id:0018    oncelik: P3  kalan sure: 0 sn)
Saniye  = 52
      proses yurutuldu. (id:0022    oncelik: P3  kalan sure: 2 sn)
      proses askiya alindi.(id:0022    oncelik: P3  kalan sure: 1 sn)
Saniye  = 53
      proses yurutuldu. (id:0002    oncelik: P3  kalan sure: 1 sn)
      proses sonlandirildi. (id:0002    oncelik: P3  kalan sure: 0 sn)
Saniye  = 54
      proses yurutuldu. (id:0014    oncelik: P3  kalan sure: 2 sn)
      proses askiya alindi.(id:0014    oncelik: P3  kalan sure: 1 sn)
Saniye  = 55
      proses yurutuldu. (id:0011    oncelik: P3  kalan sure: 1 sn)
      proses sonlandirildi. (id:0011    oncelik: P3  kalan sure: 0 sn)
Saniye  = 56
      proses yurutuldu. (id:0012    oncelik: P3  kalan sure: 1 sn)
      proses sonlandirildi. (id:0012    oncelik: P3  kalan sure: 0 sn)
```



Teşekkürler.