İşletim Sistemleri Proje Ödevi

G211210033

Havva Nur Kaymakçı -2B

G201210573

Marya KHALED MAHMOUD HIJAZI -2B

G211210577

Meys Elrim İsaelhatib - 2C

G171210041

Fatih Han Uzun - 2A

B201210005

Ali Sonat Bulut 1A

**Projenin amacı**: Bu projenin amacı, sınırlı kaynaklar içinde çalışan bir çoklu programlama sistemini simüle etmektir. Proje, aynı anda birden fazla işlemin yönetilmesi ve işlemcilerin etkin bir şekilde paylaşılmasını sağlar. İşlemleri dört farklı öncelik düzeyinde yöneten bir görevlendirici kullanır. Gerçek zamanlı ve kullanıcı işlemlerini belirli önceliklere göre sıralar ve işlemleri uygun kuyruklara yönlendirir. Proje, yazıcı, tarayıcı, modem, CD sürücüleri ve bellek gibi sınırlı kaynakları etkili bir şekilde yönetmeyi amaçlar. İşlemlere bellek tahsisi yapar ve bu bellekleri etkili bir şekilde kullanır.

**Bellek tahsis algoritmasi**

Yerleştirme Algoritmaları(Dinamik Bölmeleme):

İşletim sistemi kararıyla bellekte hangi boş bloğun hangi prosese atanacağı anlamına gelir.

En iyi ve az yer kaplayan algoritma: Best-fit algoritması(En uygun durum) algoritmasıdır Best-Fit, kullanılabilir bellek blokları arasında, en iyi (en küçük ve işlemi en iyi sığdıran) bloğu seçer.. Belleği daha etkili bir şekilde kullanabilir, parçalanmayı en aza indirebilir. Küçük işlemleri sığdırmak için daha iyi bir seçenek sunabilir.

Bundan dolayı biz de Best-fit kullanacağız.

**Prosesleri kuyruğa almak, göndermek ve bellek tahsis etme**

**Gerçek Zamanlı Proses Kuyruğu:**

Gerçek zamanlı proses kuyruğu, sistemin öncelikle gerçek zamanlı işlemlere odaklandığı bir kuyruk türüdür..İlk Gelen İlk Çalışır (FCFS) algoritması temelinde çalışır, yani gelen işlemler sırayla çalıştırılır. Gerçek zamanlı prosesler, tamamlanana kadar kesilmeden yürütülür.Bu kuyruk, özellikle zaman hassasiyeti olan uygulamalar için önemlidir ve bu işlemlerin belirlenen zaman sınırları içinde tamamlanmasını sağlar.Ayrıca Gerçek zamanlı prosesler, daha düşük öncelikle çalışan diğer proseslerden daha yüksek bir önceliğe sahiptir. Bu öncelik düzeni, kritik ve zaman duyarlı işlemlerin diğer işlemlerden önce çalıştırılmasını sağlar.

**Kullanıcı Proses Kuyruğu:**

Kullanıcı proses kuyruğu, genel kullanıcı işlemlerini yöneten bir kuyruk türüdür. Bu kuyruk, üç seviyeli bir geri beslemeli görevlendirici tarafından işletilen bir yapı içerir. Kullanıcı prosesleri, farklı öncelik seviyelerine göre gruplandırılarak sırayla çalıştırılır. Her öncelik seviyesi, belirlenen temel zamanlama kuantumu içinde işlem yapar ve ardından önceliği düşürülerek bir alt kuyruğa geçirilir. Bu şekilde, kullanıcı prosesleri arasında adil bir paylaşım sağlanır ve işlemci zamanı etkili bir şekilde yönetilir.

**Kullanılan Zamanlama Algoritmaları**

FCFS : (First-Come-First-Serve) algoritması, işlemcide çalışacak işlemlerin sırayla geliş sırasına göre sıralandığı ve ilk gelen işlemin ilk olarak işleme alındığı basit bir işlem planlama algoritmasıdır. FCFS, işlemleri bir kuyruk yapısında tutar ve kuyruktaki ilk işlemi seçerek işleme alır.

Round Robin (RR): Çoklu işlemlerin birbirleriyle sırayla ve eşit paylaşımlı bir şekilde işlemci zamanına erişmesini sağlar. Bu algoritma, işlemler arasında adil bir paylaşım sağlamak ve işlemlerin uzun süre bekletilmeden işlemciye erişmesini mümkün kılmak amacıyla kullanılır.

Çok Seviyeli Geri Besleme Kuyruğu Zamanlama Algoritması (Multilevel Feedback Queue Scheduling Algorithm): Bu algoritma, işlemleri farklı öncelik düzeylerindeki kuyruklarda sıralar ve işlemlerin önceliklerini dinamik olarak ayarlayarak işlemcilere dağıtır. Çok seviyeli geri besleme kuyruğu, hem yüksek öncelikli işlemlerin hızla tamamlanmasına izin verir hem de düşük öncelikli işlemlerin adil bir şekilde işlemcilere erişimini sağlar.

Öncelikli Zamanlama Algoritması (Priority Scheduling Algorithm): Bu algoritma, her işlemi öncelik değerine göre sıralayarak, en yüksek önceliğe sahip işlemleri önce çalıştırır. Öncelik değeri genellikle işlemi tanımlayan bir özellik veya öncelik ataması tarafından belirlenir.

**Proje İçeriği**

```
10 class Resource {
11     int printers;
12     int scanner;
13     int modem;
14     int cdDrive;
15 
16     public Resource(int printers, int scanner, int modem, int cdDrive) {
17         this.printers = printers;
18         this.scanner = scanner;
19         this.modem = modem;
20         this.cdDrive = cdDrive;
21     }
22 
23     @Override
24     public String toString() {
25         return "Resource{" +
26                 "printers=" + printers +
27                 ", scanner=" + scanner +
28                 ", modem=" + modem +
29                 ", cdDrive=" + cdDrive +
30                 '}';
31     }
32 }
```

Kaynak tahsisi için gerekli tanımlamalar yapılır ve oluşturucu yazılır.

```java
class Process {
    int arrivalTime;
    int priority;
    int executionTime;
    int memoryRequirement;
    Resource requestedResources;
    int processId;

    public Process(int arrivalTime, int priority, int executionTime, int memoryRequirement, Resource requestedResources, int processId) {
        this.arrivalTime = arrivalTime;
        this.priority = priority;
        this.executionTime = executionTime;
        this.memoryRequirement = memoryRequirement;
        this.requestedResources = requestedResources;
        this.processId = processId;
    }

    @Override
    public String toString() {
        return "Process{" +
                "processId=" + processId +
                ", arrivalTime=" + arrivalTime +
                ", priority=" + priority +
                ", executionTime=" + executionTime +
                ", memoryRequirement=" + memoryRequirement +
                ", requestedResources=" + requestedResources +
                '}';
    }
}
```

Yine ayni şekilde Proses için gerekli değişkenler yazılır ve daha sonra string formatına dönüştümek için metot yazılır.

```java
class Dispatcher {
    Queue<Process> realTimeQueue;
    Queue<Process> userQueue;
    Resource availableResources;
    int availableMemory;

    public Dispatcher(Resource availableResources, int availableMemory) {
        this.realTimeQueue = new LinkedList<>();
        this.userQueue = new LinkedList<>();
        this.availableResources = availableResources;
        this.availableMemory = availableMemory;
    }

    public void addProcess(Process process) {
        if (process.priority == 0) {
            realTimeQueue.add(process);
        } else {
            userQueue.add(process);
        }
    }

    public void runDispatcher() {
        while (!realTimeQueue.isEmpty() || !userQueue.isEmpty()) {
            if (!realTimeQueue.isEmpty()) {
                runRealTimeProcess();
            } else if (!userQueue.isEmpty()) {
                runUserProcess();
            }
        }
    }
```

Kaynakları tahsis eden metotlar yazılır.

```
private void runRealTimeProcess() {
    Process process = realTimeQueue.poll();
    System.out.println("Running real-time process: " + process);
    simulateProcessExecution(process);
}

private void runUserProcess() {
    Process process = userQueue.poll();
    System.out.println("Running user process: " + process);
    simulateProcessExecution(process);
}
```

Gerçek zamanlı proses metodu ve Kullanıcı proses metodu

```
private void simulateProcessExecution(Process process) {
    if (process.requestedResources.printers <= availableResources.printers &&
            process.requestedResources.scanner <= availableResources.scanner &&
            process.requestedResources.modem <= availableResources.modem &&
            process.requestedResources.cdDrive <= availableResources.cdDrive &&
            process.memoryRequirement <= availableMemory) {

        // Simulate process execution
        for (int i = 0; i < process.executionTime; i++) {
            System.out.println("Process " + process.processId + " is running. Time: " + i + " seconds");
        }

        System.out.println("Process completed: " + process);

        // Release resources
        availableMemory += process.memoryRequirement;
        availableResources.printers += process.requestedResources.printers;
        availableResources.scanner += process.requestedResources.scanner;
        availableResources.modem += process.requestedResources.modem;
        availableResources.cdDrive += process.requestedResources.cdDrive;

    } else {
        System.out.println("Insufficient resources for process: " + process);
    }
}
```

```
public class main {
    public static void main(String[] args) {
        // Define system resources
        Resource availableResources = new Resource(2, 1, 1, 2);
        int availableMemory = 1024;

        // Create dispatcher
        Dispatcher dispatcher = new Dispatcher(availableResources, availableMemory);

        // Read processes from the input file
        try (BufferedReader br = new BufferedReader(new FileReader("C:\\Users\\Han\\Desktop\\TEst\\giris.txt"))) {
            String line;
            int processId = 1;

            while ((line = br.readLine()) != null) {
                String[] tokens = line.split(",");
                int arrivalTime = Integer.parseInt(tokens[0].trim());
                int priority = Integer.parseInt(tokens[1].trim());
                int executionTime = Integer.parseInt(tokens[2].trim());
                int memoryRequirement = Integer.parseInt(tokens[3].trim());
                int printers = Integer.parseInt(tokens[4].trim());
                int scanner = Integer.parseInt(tokens[5].trim());
                int modem = Integer.parseInt(tokens[6].trim());
                int cdDrive = Integer.parseInt(tokens[7].trim());

                Resource requestedResources = new Resource(printers, scanner, modem, cdDrive);

                Process process = new Process(arrivalTime, priority, executionTime, memoryRequirement, requestedResources, processId);
                dispatcher.addProcess(process);

                processId++;
            }
        } catch (IOException e) {
            e.printStackTrace();
        }

        // Run the dispatcher
        dispatcher.runDispatcher();
```

Main içerisinde Kod, bir giriş dosyasındaki (giris.txt) satırları okur.Her satır, varış zamanı, öncelik, yürütme süresi vb. gibi çeşitli özelliklere sahip bir süreci temsil eder.Süreç ayrıntıları satırdan çıkarılır, uygun veri türlerine dönüştürülür ve bir Süreç nesnesi oluşturmak için kullanılır.Daha sonra her Process nesnesi addProcess yöntemi kullanılarak Dispatcher'a eklenir.

```
        // Run the dispatcher
        dispatcher.runDispatcher();
```

Dispatcher örneğinde runDispatcher yöntemi çağrılır; bu, programın asıl amacının, dağıtıcıyı kullanarak süreçlerin yürütülmesini simüle etmek olduğunu gösterir.