SAKARYA ÜNİVERSİTESİ

BİLGİSAYAR MÜHENDİSLİĞİ

İŞLETİM SİSTEMLERİ

PROJE TASARIM

Hazırlayanlar;

Mehmet Ok-G211210018

Demirhan Söylemez-G211210010

İlyas Kalfa- G211210056

Halil İbrahim Boztaş- G211210082

Ramazan Sefa Kurtuluş- G211210004

Grup47

KONU

4 Seviyeli geri beslemeli görevlendirici tasarımı.

1. **Bellek ayırma algoritmasının seçimi**

   Kullanılanilecek bellek tahsisi algoritmaları bitişik tahsis,dinamik depolama(first-fit, best-fit,worst-fit),parçalanma ve sayfalama olarak belirlendi.

   Bitişik tahsiste tahsis edilen alanların arasındaki boş alanları kullanma imkanı az olduğu ve projede verilen bellek boyutu kısıtlı olduğu için bitişik tahsis verimsiz oldu.

   Parçalanma ise belleği uygun boyutlarda bölmek için diğer algoritmalara göre daha fazla işlem gücü gerektirdiği için kullanılmadı.

   Dinamik depolama ise sistemdeki kaynak ve bellek kısıtlı olduğundan dolayı proseslerin çoğunluğu proses zamanı 20 saniyeyi aşıp silindiğinden dolayı gereksiz oldu.

   Sonuç olarak bellek ayırma algoritması için sayfalamayı uygun gördük.

2. **Görevlendirici tarafından kullanılan yapıların tanımı**

   Geri beslemeli görevlendiricinin run() metodu:

```
public void run() {
    Boolean didP1Run = false;
    Boolean didP2Run = false;
    if (!priorityFirst.isEmpty())
    {
        //1 oncelikli proses calisacak
    }
    if (!prioritySecond.isEmpty() && !didP1Run)
    {
        //2 oncelikli proses calisacak
    }
    if (!priorityThird.isEmpty() && !didP2Run && !didP1Run)
    {
        //3 oncelikli proses calisacak
    }
}
```

   Bu fonksiyon gbg'nin içinde çalışan round-robin algoritması olarak tasarlandı. Mantığı öncelikli kuyruk boşsa yada öncelikli kuyrukta çalışabilecek proses yoksa daha düşük öncelikli kuyruğu çalıştırma üzerine kuruludur.

Fonksiyon her çalışmasında eğer bir proses çalışmış ise o prosesin burs time ‘ı 1 saniye azalır.

Real time queue için run() metodu:
*Bu metot FirstComeFirstServe algoritması içine yazılmıştır.

```
public void run() {
    // fcfs algoritması

    Process curP = fcfsQueue.poll();

    curP.burst_time--;
    curP.process_status = Status.running;
    if (curP.burst_time > 0) {
        fcfsQueue.add(curP);
        curP.process_status = Status.ready;
    }
    else {
        curP.process_status=Status.terminated;
    }
}
```

Bu fonksiyonda yine gbg’nin run() metoduna benzer şekilde çalışır fakat bunda FCFS algoritması kullanıldı. Prosesin durumu ve burst time burada değişecektir. Kuantum süresi 1 saniyedir. Her proses sırasıyla 1 saniye çalışır ve kuyruğun sonuna atılır. Dolayısıyla FCFS algoritması gerçeklenmiş olacaktır.

3. **<u>Program yapısının ve bireysel modüllerin tanımı</u>**

Modüller:

- Frame.java
  - Bellekteki çerçeve yapısının gerçeklenmesi için oluşturulmuştur. 16 mb kapasitesi vardır.
- RealJobFrame.java
  - Bellekteki realTimeProcess’ler için ayrılmış alandır. Frame yapısı ile benzer özelliktedir fakat 64 mb kapasitesi vardır.
- Memory.java
  - Bellek yapısıdır. İçinde bellekBoyutu/frameBoyutu kadar frame ve 1 adet realJobFrame tutar. Frameler arraylist yapısında gerçeklenmiştir.

- GeriBeslemeliGorevlendiri.java
    - User prosesleri için yazılmış yapıdır. 3 adet farklı önceliğe sahip kuyruk içerir. Kuyruklarda round-robin algoritması gerçeklenmiştir.
- Process.java
    - Prosesler için yazılmış sınıftır. Gerçek bir proses oluşturmak yerine program içinde sanal bir proses yapısı oluşturulmuştur.
- FirstComeFirstServe.java
    - Real time prosesler için yazılmıştır. İçinde 1 adet real time proseslerin tutulduğu kuyruk vardır.
- Resource.java
    - Kaynak sınıfıdır. Sisteme hangi kaynağın ve o kaynaktan kaç adet olduğunun kaydedilmesi için yazılmıştır.

4. **<u>Görevlendiricinin tartışılması</u>**

GBG ya FCFS farketmeksizin prosesler bu iki sınıfın içindeki kuyruklara eklenir. GBG de ilgili önceliğe göre ilgili kuyruğa ekleme işlemi yapılmaktadır. FCFS de ise realTimeQueue ye ekleme yapılır.

Eğer proses sistemde bulunandan daha fazla kaynak yada bellek talep ediyorsa o proses silinir. Çalışma anında eğer proses için yeterli kaynak yada bellek yoksa o proses bekletilir ve çalışmakta olan prosesler bittiğinde yeterli bellek yada kaynak serbest kalırsa proses çalışır.

Ödevde bellek tahsisini ekleyemedik fakat kaynak tahsisi algoritması çalışmaktadır.

Eğer prosesler tahsis ettiği kaynaklardan dolatı deadlocka sebep oluyorsa kaynaklar prosese verilmeyip o proses bekletilmektedir.