**Yazılım Mühendisliği-2.Sınıf**

**Zeynep Nuriye Tekin**

**220609010**

**Kemal Çağrı Serdaroğlu**

**İşletim Sistemleri**

**Assignment 3**

**İÇİNDEKİLER:**

1. **A bölümü kodları -syf3**
2. **A kod çıktısı -syf7**
3. **B bölümü kodları -syf8**
4. **B kod çıktısı -syf13**
5. **Neden senkronizasyona ihtiyaç duyarız -syf13**
6. **Senkronizasyon algoritmasının pseudo code’u -syf14**

**A Kodu**

package application;

import java.util.concurrent.\*;

import java.util.LinkedList;

import java.util.Queue;

import java.util.Random;

//yazıcıya gonderilecek gorevleri temsil eder

class PrintJob {

int size; // in KB

public PrintJob(int size) {

this.size = size;

}

}

“PrintJob” adında bir sınıf tanımlanır; bu sınıf, yazıcıya gönderilecek belgeleri temsil eder ve bir belgenin boyutunu kilobayt cinsinden saklar. Bu sınıfın kurucu metodu, belge boyutunu parametre olarak alır ve belge oluşturulurken boyutunu belirler.

//yazıcı kuyrugunu yonetir

class PrintQueue {

private final int MAX\_BUFFER\_SIZE = 2048; // yazıcı kuyrugunun max kapasitesi:2048 KB

private final Semaphore empty;

private final Semaphore full;

private final Semaphore mutex;

private final Queue<PrintJob> buffer = new LinkedList<>();

private int currentBufferSize = 0;

public PrintQueue() {

empty = new Semaphore(MAX\_BUFFER\_SIZE);

full = new Semaphore(0);

mutex = new Semaphore(1);

}

public void addJob(PrintJob job) throws InterruptedException {

empty.acquire(job.size);

mutex.acquire();

buffer.add(job);

currentBufferSize += job.size;

System.out.println("Boyutu: " + job.size + " KB olan iş eklendi. Güncel tampon boyutu: " + currentBufferSize + " KB.");

mutex.release();

full.release(job.size);

}

public PrintJob getJob() throws InterruptedException {

full.acquire();//kuyrugun dolması beklenir

mutex.acquire();

PrintJob job = buffer.poll();

if (job != null) {

currentBufferSize -= job.size;

System.out.println("Boyutu: " + job.size + " KB olan iş işleniyor. Güncel tampon boyutu: " + currentBufferSize + " KB.");

}

mutex.release();

empty.release(job.size);

return job;

}

}

Bir yazıcı kuyruğunu yönetmek için bir sınıf olan PrintQueue'yi tanımlar. Bu sınıf, aynı anda birden fazla işin güvenli bir şekilde işlenmesini sağlamak için senkronizasyon mekanizmaları kullanır.

MAX\_BUFFER\_SIZE sabiti, yazıcı kuyruğunun maksimum kapasitesini belirler ve burada 2048 KB olarak tanımlanır.

empty, full ve mutex adında üç adet Semaphore nesnesi, kuyruğun durumunu kontrol etmek ve eşzamanlı erişimi düzenlemek için kullanılır.

buffer adında bir kuyruk yapısı, PrintJob nesnelerini saklamak için kullanılır.

currentBufferSize değişkeni, kuyruktaki toplam iş boyutunu izlemek için kullanılır.

PrintQueue sınıfının kurucu metodu, semaforları başlangıç değerleriyle başlatır ve kuyruğu boş bir şekilde oluşturur. addJob metodu, kuyruğa iş eklerken senkronizasyon sağlar. Bu metod, empty semaforunu kullanarak kuyruk doluluk durumunu kontrol eder. Bir iş eklendiğinde, mutex semaforu ile kritik bölgeye girilir, iş kuyruğa eklenir ve currentBufferSize güncellenir. Son olarak, full semaforu işin boyutuna göre serbest bırakılır.

getJob metodu ise kuyruktan iş alırken senkronizasyonu sağlar. Bu metod, full semaforunu kullanarak kuyruğun doluluk durumunu kontrol eder. Bir iş alındığında, mutex semaforu ile kritik bölgeye girilir, iş kuyruktan çıkarılır ve currentBufferSize güncellenir. Son olarak, empty semaforu işin boyutuna göre serbest bırakılır ve alınan iş geri döndürülür. Bu işlemler, kuyruğun güvenli bir şekilde işlenmesini sağlar ve eşzamanlı erişim sorunlarını önler.

//yazıcıyı simüle eder

class Printer implements Runnable {

private final PrintQueue printQueue;

public Printer(PrintQueue printQueue) {

this.printQueue = printQueue;

}

@Override

public void run() {

try {

while (true) {

PrintJob job = printQueue.getJob();

if (job != null) {

Thread.sleep(job.size \* 10); // yazdırma simülasyon süresi

System.out.println("Boyutu: " + job.size + " KB olan iş yazdırıldı.");

}

}

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

}

}

Printer adında bir sınıfı tanımlar ve bu sınıf Runnable arabirimini uygular. Printer sınıfı, yazıcı işlemlerini simüle eder. Kurucu metodunda, bir PrintQueue örneği alır ve bu örneği sınıfın özel bir değişkenine atar. run metodunda, bir while döngüsü kullanarak sürekli olarak kuyruktan iş alır ve alınan işi belirli bir süre boyunca işler. Bu süre, işin boyutuyla orantılı olarak ayarlanır. Son olarak, işin yazdırıldığına dair bir çıktı ekrana yazdırılır. Bu sınıf, gerçek bir yazıcıyı temsil etmek yerine, yazıcı kuyruğundan iş alıp bu işleri belirli bir süre boyunca işlemek için tasarlanmıştır.

class JobProducer implements Runnable {

private final PrintQueue printQueue;

private final Random random = new Random();

public JobProducer(PrintQueue printQueue) {

this.printQueue = printQueue;

}

@Override

public void run() {

try {

while (true) {

int jobSize = random.nextInt(463) + 50; //50 ila 512 KB arasında rastgele boyut

PrintJob job = new PrintJob(jobSize);

printQueue.addJob(job);

Thread.sleep(random.nextInt(1000)); //Bir sonraki işin eklenmesinden önce rastgele gecikme

}

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

}

}

JobProducer adında bir sınıfı tanımlanır ve bu sınıf Runnable arabirimini uygular. JobProducer sınıfı, yazıcı kuyruğuna işler ekleyen bir üreticiyi temsil eder. Kurucu metodunda, bir PrintQueue örneği alır ve bu örneği sınıfın özel bir değişkenine atar. run metodunda, bir while döngüsü kullanarak sürekli olarak yeni işler üretir ve bu işleri yazıcı kuyruğuna ekler. Her bir işin boyutu, 50 ile 512 KB arasında rastgele olarak belirlenir. Ayrıca, her işin eklenmesi arasında rastgele bir gecikme oluşturmak için Thread.sleep yöntemi kullanılır. Bu sınıf, yazıcıya iş gönderen bir arka plan işleyici olarak tasarlanmıştır ve yazıcı işlemlerinin sürekli olarak devam etmesini sağlar.

public class odev3\_A {

public static void main(String[] args) {

PrintQueue printQueue = new PrintQueue();//yazıcı kuyrugunu yonetir

Printer printer = new Printer(printQueue);

Thread printerThread = new Thread(printer);

JobProducer producer = new JobProducer(printQueue);

Thread producerThread = new Thread(producer);

printerThread.start(); //yaziciyi simule eden thread olusturur

producerThread.start(); //yazıcıya is ureten thread olusturur

}

}

`odev3\_A` adında bir genel sınıf tanımlanır. `main` metodu, programın giriş noktasını oluşturur. İlk olarak, `PrintQueue` sınıfından bir örnek oluşturulur ve bu örnek, bir yazıcı kuyruğunu yönetmek için kullanılır. Ardından, bu kuyruğu parametre olarak alarak bir `Printer` ve bir `JobProducer` örneği oluşturulur. `Printer` örneği, yazıcının gerçekleştirilmesini simüle eden bir iş parçacığını temsil ederken, `JobProducer` örneği, yazıcı kuyruğuna iş gönderen bir arka plan işleyicisini temsil eder.

Daha sonra, `Printer` ve `JobProducer` örnekleri, ayrı iş parçacıklarında (`Thread`), `printerThread` ve `producerThread` olarak başlatılır. Bu iş parçacıkları, yazıcı işlemlerini ve iş üretimini aynı anda gerçekleştirerek paralel çalışmayı sağlar. `printerThread` yazdırma işlemlerini simüle ederken, `producerThread` iş üretimini gerçekleştirir. Bu şekilde, yazıcı işlemleri ve iş üretimi arka planda devam eder ve ana program akışı kesintiye uğramadan çalışır.

**Çıktı:**

Boyutu: 114 KB olan iş eklendi. Güncel tampon boyutu: 114 KB.

Boyutu: 114 KB olan iş işleniyor. Güncel tampon boyutu: 0 KB.

Boyutu: 175 KB olan iş eklendi. Güncel tampon boyutu: 175 KB.

Boyutu: 114 KB olan iş yazdırıldı.

Boyutu: 175 KB olan iş işleniyor. Güncel tampon boyutu: 0 KB.

Boyutu: 355 KB olan iş eklendi. Güncel tampon boyutu: 355 KB.

Boyutu: 442 KB olan iş eklendi. Güncel tampon boyutu: 797 KB.

Boyutu: 458 KB olan iş eklendi. Güncel tampon boyutu: 1255 KB.

Boyutu: 175 KB olan iş yazdırıldı.

Boyutu: 355 KB olan iş işleniyor. Güncel tampon boyutu: 900 KB.

Boyutu: 482 KB olan iş eklendi. Güncel tampon boyutu: 1382 KB.

Boyutu: 417 KB olan iş eklendi. Güncel tampon boyutu: 1799 KB.

Boyutu: 242 KB olan iş eklendi. Güncel tampon boyutu: 2041 KB.

Boyutu: 355 KB olan iş yazdırıldı.

Boyutu: 442 KB olan iş işleniyor. Güncel tampon boyutu: 1599 KB.

Boyutu: 409 KB olan iş eklendi. Güncel tampon boyutu: 2008 KB.

Boyutu: 442 KB olan iş yazdırıldı.

Boyutu: 458 KB olan iş işleniyor. Güncel tampon boyutu: 1550 KB.

Boyutu: 330 KB olan iş eklendi. Güncel tampon boyutu: 1880 KB.

Boyutu: 458 KB olan iş yazdırıldı.

Boyutu: 482 KB olan iş işleniyor. Güncel tampon boyutu: 1398 KB.

Boyutu: 224 KB olan iş eklendi. Güncel tampon boyutu: 1622 KB.

Boyutu: 396 KB olan iş eklendi. Güncel tampon boyutu: 2018 KB.

Boyutu: 482 KB olan iş yazdırıldı.

Boyutu: 417 KB olan iş işleniyor. Güncel tampon boyutu: 1601 KB.

Boyutu: 393 KB olan iş eklendi. Güncel tampon boyutu: 1994 KB.

Boyutu: 417 KB olan iş yazdırıldı.

**B Kodu:**

package application;

import java.util.concurrent.locks.\*;

import java.util.Queue;

import java.util.LinkedList;

import java.util.Random;

public class odev3\_B {

// PrintJob sınıfı

static class PrintJob {

int size; // KB cinsinden

public PrintJob(int size) {

this.size = size;

}

}

odev3\_B adında bir Java sınıfı tanımlanır. Bu sınıf, bir yazıcı iş parçacığı kuyruğunu yönetmek için kullanılan bir mekanizmayı içerir. İlk olarak, PrintJob adında bir iç içe sınıf tanımlanır. Bu sınıf, yazdırılacak işleri temsil eder ve işlerin boyutunu kilobayt (KB) cinsinden saklar. Ardından, odev3\_B sınıfı, yazıcı işlerini kuyrukta yönetmek için gerekli olan semaforlar, kilitler ve kuyruk yapısını tanımlar. Kuyrukta işlerin boyutlarına göre sıralı bir şekilde eklenmesini ve çıkarılmasını sağlamak için bir Queue uygulaması olan LinkedList kullanılır. Bu sınıf ayrıca, mevcut kuyruk boyutunu izlemek için bir değişken (currentBufferSize) ve bu boyutu kontrol etmek için bir sabit (MAX\_BUFFER\_SIZE) içerir. Bu sabit, kuyruğun maksimum boyutunu KB cinsinden belirtir.

// PrintQueue sınıfı

static class PrintQueue {

private final int MAX\_BUFFER\_SIZE = 2048; // Maksimum tampon boyutu: 2048 KB

private final Lock lock = new ReentrantLock(); // Kilitleme mekanizması için kullanılacak lock

private final Condition notFull = lock.newCondition(); // Tampon dolu olmadığında beklemek için

private final Condition notEmpty = lock.newCondition(); // Tampon boş olmadığında beklemek için

private final Queue<PrintJob> buffer = new LinkedList<>(); // İşlerin tutulacağı kuyruk

private int currentBufferSize = 0; // Mevcut tampon boyutu

public void addJob(PrintJob job) throws InterruptedException {

lock.lock(); // Kilidi al

try {

// Tampon dolu olduğunda beklenir

while (currentBufferSize + job.size > MAX\_BUFFER\_SIZE) {

notFull.await();

}

// İş tampona eklendi

buffer.add(job);

currentBufferSize += job.size;

// Eklenen işin boyutu ve mevcut tampon boyutu yazdırılır

System.out.println("Eklenen işin boyutu: " + job.size + " KB. Mevcut tampon boyutu: " + currentBufferSize + " KB.");

// Tamponun artık boş olmadığına dair sinyal gönderilir

notEmpty.signal();

} finally {

lock.unlock(); // Kilidi serbest bırak

}

}

`PrintQueue` adında bir iç içe sınıfı tanımlanır. Bu sınıf, yazıcı işlerini yönetmek için bir kuyruk yapısı kullanır. İşleri kuyruğa eklemek ve kuyruktan çıkarmak için senkronize bir şekilde çalışır.

Sınıfın başında, maksimum tampon boyutunu (`MAX\_BUFFER\_SIZE`), kilitleme mekanizması için bir kilit nesnesi (`lock`), tamponun dolu olmadığında beklemek için bir koşul değişkeni (`notFull`), tamponun boş olmadığında beklemek için bir koşul değişkeni (`notEmpty`), işlerin tutulacağı bir kuyruk (`buffer`), ve mevcut tampon boyutunu tutmak için bir değişken (`currentBufferSize`) tanımlanır.

`addJob` metodu, iş eklemek için kullanılır. Bu metod, iş eklemek için kuyruğun kilidini alır (`lock.lock()`). Ardından, tampon dolu olduğunda beklemek için bir döngü kullanır. İş tampona eklenir ve tampon boyutu güncellenir. Eklenen işin boyutu ve mevcut tampon boyutu yazdırılır. Son olarak, tamponun artık boş olmadığına dair bir sinyal gönderilir (`notEmpty.signal()`). Her zaman kilidi serbest bırakmak için `finally` bloğunda kilidi serbest bırakır (`lock.unlock()`).

public PrintJob getJob() throws InterruptedException {

lock.lock(); // Kilidi al

try {

// Tampon boş olduğunda beklenir

while (buffer.isEmpty()) {

notEmpty.await();

}

// Bir iş alınır

PrintJob job = buffer.poll();

if (job != null) {

currentBufferSize -= job.size;

// İşlenen işin boyutu ve mevcut tampon boyutu yazdırılır

System.out.println("İşlenen işin boyutu: " + job.size + " KB. Mevcut tampon boyutu: " + currentBufferSize + " KB.");

// Tamponun artık dolu olmadığına dair sinyal gönderilir

notFull.signal();

}

return job;

} finally {

lock.unlock(); // Kilidi serbest bırak

}

}

}

getJob adında bir metod tanımlanır. Bu metod, kuyruktan iş almak için kullanılır. Metodun başında, kilit alınır (lock.lock()). Ardından, tampon boş olduğunda beklemek için bir döngü kullanılır. Kuyruk boş ise, notEmpty.await() ifadesi ile beklenir. Döngüden çıkıldığında, bir iş kuyruktan alınır (buffer.poll()). Eğer alınan iş null değilse, yani kuyruk boş değilse, işlenen işin boyutu kadar mevcut tampon boyutundan düşülür ve bu bilgiler yazdırılır. Son olarak, tamponun artık dolu olmadığına dair bir sinyal gönderilir (notFull.signal()). Her zaman kilidi serbest bırakmak için finally bloğunda kilidi serbest bırakır (lock.unlock()).

// Printer sınıfı

static class Printer implements Runnable {

private final PrintQueue printQueue;

public Printer(PrintQueue printQueue) {

this.printQueue = printQueue;

}

@Override

public void run() {

try {

while (true) {

// Yazıcı kuyruğundan bir iş alınır

PrintJob job = printQueue.getJob();

if (job != null) {

// İşin boyutu kadar beklenir (yazdırma simülasyonu)

Thread.sleep(job.size \* 10);

// Yazdırılan işin boyutu yazdırılır

System.out.println("Yazdırılan işin boyutu: " + job.size + " KB.");

}

}

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

}

}

`Printer` adında bir iç içe sınıf tanımlanır. Bu sınıf, bir yazıcıyı temsil eder ve `Runnable` arabirimini uygular, böylece bir iş parçacığı olarak çalıştırılabilir.

Yapıcı metodunda, bir `PrintQueue` örneği alır ve bu örneği sınıf içindeki bir değişkende saklar.

`run` metodu, yazıcı iş parçacığının ana işleyişini belirtir. Sonsuz bir döngü içinde, yazıcı kuyruğundan bir iş alır (`printQueue.getJob()`). Eğer alınan iş null değilse, yani kuyruk boş değilse, yazdırma simülasyonunu gerçekleştirmek için işin boyutu kadar bekler (`Thread.sleep(job.size \* 10)`) ve ardından yazdırılan işin boyutunu yazdırır (`System.out.println("Yazdırılan işin boyutu: " + job.size + " KB.")`).

Metodun sonunda `InterruptedException` yakalanır ve iş parçacığına kesme işareti gönderilir (`Thread.currentThread().interrupt()`). Bu, iş parçacığının uykuda olması durumunda bir kesme işareti aldığında iş parçacığının uyandırılmasını sağlar.

// JobProducer sınıfı

static class JobProducer implements Runnable {

private final PrintQueue printQueue;

private final Random random = new Random();

public JobProducer(PrintQueue printQueue) {

this.printQueue = printQueue;

}

@Override

public void run() {

try {

while (true) {

// Rasgele bir iş boyutu üretilir (50-512 KB arası)

int jobSize = random.nextInt(463) + 50;

// Üretilen iş kuyruğa eklenir

PrintJob job = new PrintJob(jobSize);

printQueue.addJob(job);

// Rasgele bir bekleme süresi eklenir (0-1000 ms arası)

Thread.sleep(random.nextInt(1000));

}

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

}

}

JobProducer sınıfı, yazıcı kuyruğuna yazdırma işleri eklemek için kullanılacak bir iş parçacığıdır. Sınıf, bir PrintQueue nesnesi ve rastgele iş boyutları ve bekleme süreleri üretmek için bir Random nesnesi kullanır. Yapıcı metodunda, PrintQueue nesnesi sınıfın printQueue değişkenine atanır. run yöntemi, sınıfın bir iş parçacığı olarak çalıştırıldığında sürekli olarak çalışacak olan kodu içerir. Bu kod, sonsuz bir döngü içinde çalışarak, 50 ile 512 KB arasında rastgele bir iş boyutu oluşturur, bu boyutla yeni bir PrintJob nesnesi yaratır ve bu işi yazıcı kuyruğuna ekler. İşin kuyruğa eklenmesinden sonra, 0 ile 1000 milisaniye arasında rastgele bir süre beklenir. Döngü, InterruptedException yakalanana kadar devam eder; bu durumda iş parçacığı durdurulur

// Ana program

public static void main(String[] args) {

// Yazıcı kuyruğu oluşturulur

PrintQueue printQueue = new PrintQueue();

// Yazıcı ve iş üretici thread'ler oluşturulur

Printer printer = new Printer(printQueue);

Thread printerThread = new Thread(printer);

JobProducer producer = new JobProducer(printQueue);

Thread producerThread = new Thread(producer);

// Thread'ler başlatılır

printerThread.start();

producerThread.start();

// Opsiyonel olarak, thread'lerin bitmesini beklemek için kullanılabilir

try {

printerThread.join();

producerThread.join();

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

}

}

main yöntemi, programın başlangıç noktasını belirler ve yazıcı kuyruğu simülasyonunun çalıştırılmasını sağlar. İlk olarak, bir PrintQueue nesnesi oluşturulur. Ardından, bir Printer ve bir JobProducer nesnesi yaratılır ve bu nesnelerden Thread nesneleri oluşturulur (printerThread ve producerThread). printerThread, yazdırma işlemlerini simüle ederken producerThread, yazdırılacak işleri üretir. Bu iki iş parçacığı başlatılır (start yöntemi ile). Opsiyonel olarak, join yöntemi kullanılarak ana iş parçacığı, printerThread ve producerThread iş parçacıklarının bitmesini bekler. Eğer bu iş parçacıkları kesilirse (InterruptedException), ana iş parçacığı da kesilir ve durdurulur. Bu yapı, işlerin yazıcı kuyruğuna eklenmesi ve yazdırılması işlemlerinin senkronize ve düzgün bir şekilde gerçekleştirilmesini sağlar.

Çıktı:

Eklenen işin boyutu: 185 KB. Mevcut tampon boyutu: 185 KB.

İşlenen işin boyutu: 185 KB. Mevcut tampon boyutu: 0 KB.

Eklenen işin boyutu: 499 KB. Mevcut tampon boyutu: 499 KB.

Eklenen işin boyutu: 481 KB. Mevcut tampon boyutu: 980 KB.

Eklenen işin boyutu: 209 KB. Mevcut tampon boyutu: 1189 KB.

Yazdırılan işin boyutu: 185 KB.

İşlenen işin boyutu: 499 KB. Mevcut tampon boyutu: 690 KB.

Eklenen işin boyutu: 77 KB. Mevcut tampon boyutu: 767 KB.

Eklenen işin boyutu: 298 KB. Mevcut tampon boyutu: 1065 KB.

Eklenen işin boyutu: 119 KB. Mevcut tampon boyutu: 1184 KB.

Eklenen işin boyutu: 356 KB. Mevcut tampon boyutu: 1540 KB.

Eklenen işin boyutu: 282 KB. Mevcut tampon boyutu: 1822 KB.

Yazdırılan işin boyutu: 499 KB.

İşlenen işin boyutu: 481 KB. Mevcut tampon boyutu: 1341 KB.

Eklenen işin boyutu: 300 KB. Mevcut tampon boyutu: 1641 KB.

Eklenen işin boyutu: 295 KB. Mevcut tampon boyutu: 1936 KB.

Yazdırılan işin boyutu: 481 KB.

İşlenen işin boyutu: 209 KB. Mevcut tampon boyutu: 1727 KB.

Eklenen işin boyutu: 242 KB. Mevcut tampon boyutu: 1969 KB.

Yazdırılan işin boyutu: 209 KB.

İşlenen işin boyutu: 77 KB. Mevcut tampon boyutu: 1892 KB.

Yazdırılan işin boyutu: 77 KB.

**Senkronizasyona İhtiyaç Duyulma Sebebi**

Senkronizasyona ihtiyaç duyulmasının ana nedeni, paylaşılan kaynaklara (yazıcı kuyruğu) aynı anda birden fazla işlemci (thread) tarafından erişilmesini kontrol etmektir.

A bölümü için yazılan kodda, Semaphore ve Mutex gibi senkronizasyon araçları kullanılarak senkronizasyon sağlanmıştır. Bu mekanizmalar, yazıcı kuyruğuna erişimi sıraya koymak ve paylaşılan değişkenlerin güvenli bir şekilde güncellenmesini sağlamak için kullanılır.

**Senkronizasyon Algoritmasının Pseudocode**

**initialize:**

semaphore empty = new Semaphore(MAX\_SIZE);

semaphore full = new Semaphore(0);

semaphore mutex = new Semaphore(1);

int currentBufferSize = 0;

**addJob(job):**

empty.acquire(job.size); // Bekle, boş alan var mı?

mutex.acquire(); // Kritik bölgeye giriş yap

buffer.add(job); // İşi kuyruğa ekle

currentBufferSize += job.size; // Kuyruk boyutunu güncelle

mutex.release(); // Kritik bölgeden çıkış yap

full.release(job.size); // İşaretçiyi artır, tampon dolu

**getJob():**

full.acquire(); // Bekle, kuyruk dolu mu?

mutex.acquire(); // Kritik bölgeye giriş yap

PrintJob job = buffer.poll(); // İşi kuyruktan al

currentBufferSize -= job.size; // Kuyruk boyutunu güncelle

mutex.release(); // Kritik bölgeden çıkış yap

empty.release(job.size); // İşaretçiyi artır, tampon boş

return job; // İşi geri döndür