TUGAS SISTEM TERDISTRIBUSI
MULTYTHREAD

EKO SURIPTO P.
33301

Konsep thread

Sebuah proses adalah suatu program yang sedang dieksekusi. Proses lebih dari sebuah kode program tetapi juga mencakup program counter, stack, dan sebuah data section. seiring berjalannya waktu dan tuntutan teknologi ternyata ditemukan kelemahan yang sebenarnya bisa diminimalisir pada proses. Untuk itulah diciptakan thread yang merupakan cara dari komputer untuk menjalankan dua atau lebih task dalam waktu bersamaan, sedangkan multithreading adalah cara komputer untuk membagi-bagi pekerjaan yang dikerjakan sebagian-sebagian dengan cepat sehingga menimbulkan efek seperti menjalakan beberapa task secara bersamaan walaupun otaknya hanya satu.

Multiprocessing merupakan penggunaan dua atau lebih CPU dalam sebuah sistem komputer. Multitasking merupakan metode untuk menjalankan lebih dari satu proses dimana terjadi pembagian sumberdaya seperti CPU. Sedangkan multithreading adalah cara pengeksekusian yang mengizinkan beberapa thread terjadi dalam sebuah proses, saling berbagi sumber daya tetapi dapat dijalankan secara independen.

Keuntungan dari sistem yang menerapkan multithreading dapat kita kategorikan menjadi 4 bagian:

1. **Responsif.** Aplikasi interaktif menjadi tetap responsif meskipun sebagian dari program sedang diblok atau melakukan operasi lain yang panjang. Umpamanya, sebuah thread dari web browser dapat melayani permintaan pengguna sementara thread yang lain berusaha menampilkan gambar.

2. **Berbagi sumber daya.** Beberapa thread yang melakukan proses yang sama akan berbagi sumber daya. Keuntungannya adalah mengizinkan sebuah aplikasi untuk mempunyai beberapa thread yang berbeda dalam lokasi memori yang sama.

3. **Ekonomis**. Pembuatan sebuah proses memerlukan pengalokasian memori dan sumber daya. Alternatifnya adalah dengan menggunakan thread, karena thread membagi memori dan sumber daya yang dimilikinya sehingga lebih ekonomis untuk membuat thread dan context switching thread. Akan susah mengukur perbedaan waktu antara thread dan switch, tetapi secara umum pembuatan dan pengaturan proses akan memakan waktu lebih lama dibandingkan dengan thread. Pada Solaris, pembuatan proses memakan waktu 30 kali lebih lama dibandingkan pembuatan thread sedangkan proses context switch 5 kali lebih lama dibandingkan context switching thread.

4. **Utilisasi arsitektur multiprosesor**. Keuntungan dari multithreading dapat sangat meningkat pada arsitektur multiprosesor, dimana setiap thread dapat berjalan secara

paralel di atas procesor yang berbeda. Pada arsitektur processor tunggal, CPU menjalankan setiap thread secara bergantian tetapi hal ini berlangsung sangat cepat sehingga menciptakan ilusi paralel, tetapi pada kenyataanya hanya satu thread yang dijalankan CPU pada satu-satuan waktu.

Thread Java

Suatu proses dikontrol oleh paling sedikit satu thread. Namun, sebagian besar proses yang ada sekarang biasanya dijalankan oleh beberapa buah thread. Multithreading adalah sebuah mekanisme di mana dalam suatu proses, ada beberapa thread yang mengerjakan tugasnya masing-masing pada waktu yang bersamaan. Contohnya, sebuah web browser harus menampilkan sebuah halaman yang memuat banyak gambar. Pada program yang single-threaded, hanya ada satu thread untuk mengatur suatu gambar, lalu jika gambar itu telah ditampilkan, barulah gambar lain bisa diproses. Dengan multithreading, proses bisa dilakukan lebih cepat jika ada thread yang menampilkan gambar pertama, lalu thread lain untuk menampilkan gambar kedua, dan seterusnya, di mana thread-thread tersebut berjalan secara paralel.

Saat sebuah program Java dieksekusi, yaitu saat main() dijalankan, ada sebuah thread utama yang bekerja. Java adalah bahasa pemrograman yang mendukung adanya pembentukan thread tambahan selain thread utama tersebut. Thread dalam Java diatur oleh Java Virtual Machine(JVM) sehingga sulit untuk menentukan apakah thread Java berada di user-level atau kernel-level.

Suatu thread bisa berada pada salah satu dari status berikut:

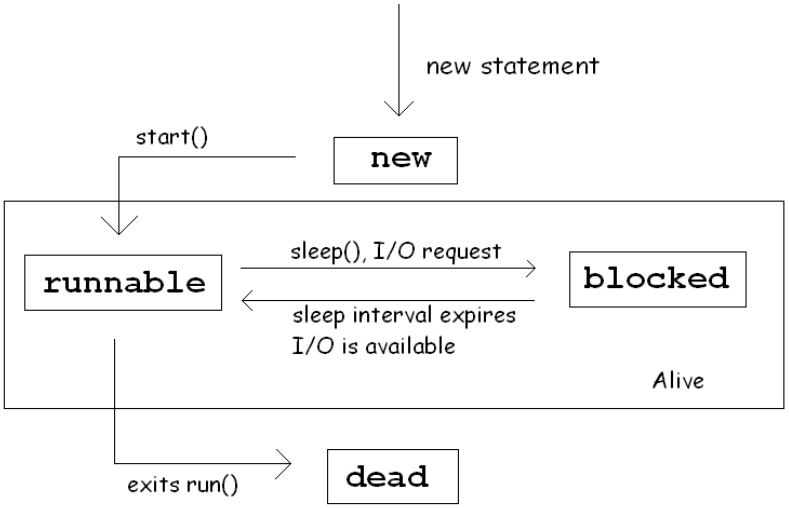
1. **New.** Thread yang berada di status ini adalah objek dari kelas Thread yang baru dibuat, yaitu saat instansiasi objek dengan statement new. Saat thread berada di status new, belum ada sumber daya yang dialokasikan, sehingga thread belum bisa menjalankan perintah apapun.

2. **Runnable.** Agar thread bisa menjalankan tugasnya, method start() dari kelas Thread harus dipanggil. Ada dua hal yang terjadi saat pemanggilan method start(), yaitu alokasi memori untuk thread yang dibuat dan pemanggilan method run(). Saat method run() dipanggil, status thread berubah menjadi runnable, artinya thread tersebut sudah memenuhi syarat untuk dijalankan oleh JVM. Thread yang sedang berjalan juga berada di status runnable.

3. **Blocked**. Sebuah thread dikatakan berstatus blocked atau terhalang jika terjadi blocking statement, misalnya pemanggilan method sleep(). sleep() adalah suatu method yang menerima argumen bertipe integer dalam bentuk milisekon. Argumen tersebut menunjukkan seberapa lama thread akan "tidur". Selain sleep(), dulunya dikenal method suspend(), tetapi sudah disarankan untuk tidak digunakan lagi karena mengakibatkan terjadinya deadlock. Di samping blocking statement, adanya interupsi M/K juga dapat menyebabkan thread menjadi blocked. Thread akan menjadi runnable kembali jika interval method sleep()-nya sudah berakhir, atau pemanggilan method

resume() jika untuk menghalangi thread tadi digunakan method suspend() atau M/K sudah tersedia lagi.

4. **Dead.** Sebuah thread berada di status dead bila telah keluar dari method run(). Hal ini bisa terjadi karena thread tersebut memang telah menyelesaikan pekerjaannya di method run(), maupun karena adanya pembatalan thread. Status jelas dari sebuah thread tidak dapat diketahui, tetapi method isAlive() mengembalikan nilai boolean untuk mengetahui apakah thread tersebut dead atau tidak.



Gambar 1. Status Thread

Pembentukan Thread
Ada dua cara untuk membuat thread di program Java, yaitu:

1. Implements interface Runnable.
Source code:

```
class Thread01{
        static public void main(String[] args){
                Thread myThreadA = new Thread(new MyThread(),"threadA");
                Thread myThreadB = new Thread(new MyThread(),"threadB");

                myThreadA.start();
                myThreadB.start();

                try{
                        Thread.currentThread().sleep(1000);
                }
```

```
                catch(InterruptedException e){
                }

                System.out.println(Thread.currentThread());
        }
}

//=====================================================//

class DoNothing{
}

//=====================================================//

class MyThread extends DoNothing implements Runnable{
        public void run(){
                System.out.println(Thread.currentThread());
        }
}
```

Hasil running:

```
Thread[threadA,5,main]
Thread[threadB,5,main]
Thread[main,5,main]
```

Penjelasan:

Terlihat pada source diatas class MyThread mengimplementasikan sebuah interface, Runnable. Interface Runnable didefinisikan sebagai berikut:

```
public interface Runnable {
        public abstract void run();
}
```

Sehingga dalam class tersebut harus di definisikan sebuah method run().

2. Extends kelas Thread.

Source code:

```
class Thread02{
        static public void main(String[] args){
                Thread myThreadA = new Thread(new MyThread(),"threadA");
                Thread myThreadB = new Thread(new MyThread(),"threadB");

                myThreadA.start();
                myThreadB.start();

                try{
                        Thread.currentThread().sleep(1000);
                }
```

```
                catch(InterruptedException e){}

                System.out.println(Thread.currentThread());

        }
}

//=====================================================//

class MyThread extends Thread{
        public void run(){
                System.out.println(Thread.currentThread());
        }
}
```

Hasil running:

```
Thread[threadA,5,main]
Thread[threadB,5,main]
Thread[main,5,main]
```

Source code 2:

```
class Thread06{
        static public void main(String[] args){
                Thread myThreadA = new MyThread();
                Thread myThreadB = new MyThread();

                myThreadA.start();
                myThreadB.start();

                try{
                        Thread.currentThread().sleep(1000);
                }catch(InterruptedException e){}

                System.out.println(Thread.currentThread());
         }

}

//=====================================================//

class MyThread extends Thread{
        public void run(){
                System.out.println(Thread.currentThread());
        }
}
```

Hasil running:

Thread[Thread-1,5,main]
Thread[Thread-2,5,main]
Thread[main,5,main]

Penjelasan:

Cara kedua ini menggunakan pewarisan dari kelas thread terhadap kelas MyThread. Kelas Thread secara implisit juga meng- implements interface Runnable. Oleh karena itu, setiap kelas yang diturunkan dari kelas Thread juga harus mendefinisikan method run(). Jadi, secara tidak langsung kelas yang meng-extend kelas thread juga harus mengimplementasikan method run() yang terdapat dalam interface Runnable. Perbedaan antara source1 dan 2 hanya terletak pada cara pendefinisian objek threadnya. Efek yang ditimbulkan dari perbedaan tersebut akan dijelaskan selanjutnya.

Penjelasan umum serta perbandingan ketiga contoh diatas:

Dari contoh diatas terlihat 2 buah cara pembentukan thread seperti dalam pembagiannya, yaitu dengan mengimplementasikan interface Runnable dan mewariskan kelas thread pada suatu kelas. Konsep pewarisan dalam Java tidak mendukung multiple inheritance. Jika sebuah kelas sudah meng- extends suatu kelas lain, maka kelas tersebut tidak lagi bisa meng- extends kelas Thread. Oleh karena itu, cara kedua, yaitu meng-implements interface Runnable, lebih umum digunakan, karena kita bisa meng-implements dari banyak kelas sekaligus.

Dari contoh diatas terlihat juga ada dua cara pembentukan thread objek, yaitu

Thread myThreadA = new Thread(new MyThread(),"threadA");

Thread myThreadA = new MyThread();

Perbedaan cara penginstansian objek ini terletak pada perbedaan akses yang dimiliki oleh kelas-kelas tersebut. Efek yang ditimbulkan dari perbedaan tersebut tampak pada hasil running programnya.

Thread[threadA,5,main]

Thread[Thread-1,5,main]

Perbedaaanya hanya terletak pada nama thread objek yang terbentuk. Cara pertama memberikan nama pada objek secara langsung melalui parameter contuctornya, yaitu “threadA”. Sedangkan pada cara kedua penamaan thread menggunaan penamaan default, sehingga nama thread tersebut adalah Thread-1.

Hasil running program yang ditampilkan tersebut merupakan nilai kembalian dari method Thread.currentThread(). Format kembaliajn tersebut adalah:

Thread[nama_thread, prioritas, group].

Kemudian, dalam source jugab terdapat baris

myThreadA.start();
myThreadB.start();

baris tersebut memrupakan perintah untuk memulai suatu thread dengan memanggil method start(). Ketika terjadi pemanggilan method start(), thread yang dibuat akan langsung mengerjakan baris-baris perintah yang ada di method run(). Jika run() dipanggil secara langsung tanpa melalui

start(), perintah yang ada di dalam method run() tersebut akan tetap dikerjakan, hanya saja yang mengerjakannya bukanlah thread yang dibuat tadi, melainkan thread utama.

Contoh lain yang lebih real.

Source code:

```
class Thread03{
        static public void main(String[] args){

                Thread threadA = new Thread(new MyThread(),"thrdA");
                Thread threadB = new Thread(new MyThread(),"threadB");

                threadA.start();
                threadB.start();

                try{
                        Thread.currentThread().sleep(1000);
                }
                catch(InterruptedException e){}

        }//end main
}//end class Thread03

//=========================================================//

class MyThread implements Runnable{
        public void run(){
                for(int cnt = 0; cnt < 5; cnt++){
                   System.out.println(Thread.currentThread() + " cnt is " + cnt);
                   Try{
                        Thread.currentThread().sleep((int)(Math.random() * 100));
                   }
                   catch(InterruptedException e){}
                }
        }
}
```

Hasil running:

```
Thread[thrdA,5,main] cnt is 0
Thread[threadB,5,main] cnt is 0
Thread[threadB,5,main] cnt is 1
Thread[thrdA,5,main] cnt is 1
Thread[threadB,5,main] cnt is 2
Thread[thrdA,5,main] cnt is 2
Thread[thrdA,5,main] cnt is 3
Thread[thrdA,5,main] cnt is 4
```

```
Thread[threadB,5,main] cnt is 3
Thread[threadB,5,main] cnt is 4
```

Penjelasan:

Program diatas secara umum sama saja dengan 3 program sebelumnya, yang membedakan hanya baris-baris dalam method run(). Hal tersebut dimaksudkan untuk memperlihatkan cara kerja thread-threat tersebut. Objek yang diinstant dalam program saling independen dan memulai mereka running asynchronously.

Tiap thread melakukan pencacahan/count masing-masing dari 0 sampai 4 dan menampilkan hasil pencacahannya tersebut. Tiap thread juga akan mengalami sleep diantara pencacahan tersebut dengan waktu yang random. Dan karena itu output tiap running selalu berbeda karena menggunakan bilangan random. Kadang waktu sleepnya hampir sama sehingga terlihat seolah-olah bergantian, namun pada salah satu hasil running diatas memperlihatkan thrdA secara berturut-turut melakukan 3 kali cacah. Ha,l tersebut terjadi akibat waktu sleep yang terjadi pada threadB jauh lebih lama dibanding thrdA.

Contoh yang lebih kompleks

Source code:

```
class Thread04{
        static public void main(String[] args){
                Thread threadA = new Thread(new OneThreadClass(),"thrdA");
                Thread threadB = new Thread(new OneThreadClass(),"threadB");

                Thread Xthread = new Thread(new
                        AnotherThreadClass(),"Xthrd");
                Thread Ythread = new Thread(
                        new AnotherThreadClass(),"Ythread");

                threadA.start();
                threadB.start();
                Xthread.start();
                Ythread.start();

                try{
                        Thread.currentThread().sleep(1000);
                }
                catch(InterruptedException e){}
        }
}

//=====================================================//

class OneThreadClass implements Runnable{
        public void run(){
                for(int cnt = 0; cnt < 5; cnt++){
```

```
                    System.out.println(Thread.currentThread() + " cnt is " + cnt);

                    try{
                        Thread.currentThread().sleep((int)(Math.random() * 100));
                    }
                    catch(InterruptedException e){}
                }
            }
        }
        //=====================================================//

        class AnotherThreadClass implements Runnable{
            public void run(){
                for(int cnt = 0; cnt < 5; cnt++){
                    System.out.println("The actual cnt is " + cnt + " " +
                        Thread.currentThread());
                    try{
                        Thread.currentThread().sleep((int)(Math.random() * 100));
                    }
                    catch(InterruptedException e){}
                }
            }
        }
```

Hasil running:

```
The actual cnt is 0 Thread[Ythread,5,main]
Thread[threadB,5,main] cnt is 0
Thread[thrdA,5,main] cnt is 0
The actual cnt is 0 Thread[Xthrd,5,main]
Thread[thrdA,5,main] cnt is 1
Thread[thrdA,5,main] cnt is 2
Thread[threadB,5,main] cnt is 1
Thread[thrdA,5,main] cnt is 3
The actual cnt is 1 Thread[Ythread,5,main]
The actual cnt is 1 Thread[Xthrd,5,main]
The actual cnt is 2 Thread[Ythread,5,main]
Thread[thrdA,5,main] cnt is 4
The actual cnt is 3 Thread[Ythread,5,main]
Thread[threadB,5,main] cnt is 2
The actual cnt is 2 Thread[Xthrd,5,main]
The actual cnt is 4 Thread[Ythread,5,main]
Thread[threadB,5,main] cnt is 3
The actual cnt is 3 Thread[Xthrd,5,main]
Thread[threadB,5,main] cnt is 4
The actual cnt is 4 Thread[Xthrd,5,main]
```

Penjelasan:
Pada daasarnya program diatas hampir sama dengan program sebelumnya. Yang membedakan hanya terdapat pada kelas thread class yang digunakan ada 2 yang isinya pada dasarnya sama juga, yaitu melakukan pencacahan dari 0-4 dan kemudian menampilkan hasil pencacahan. Serta terdapat juga sleep diantara pencacahan tersebut. Objek thread yang terlibat dalam program ini juga lebih banyak, yaitu 4 objek.

Sinkronisasi thread
Source code:

```
class Synch01{
        static QueueManager queueManager = new QueueManager();
        static boolean running = true;

        public static void main(String[] args){
                Thread producer = new Producer();
                Thread consumer = new Consumer();
                producer.start();
                consumer.start();

                try{ //delay two seconds
                        Thread.currentThread().sleep(2000);
                }
                catch(InterruptedException e){};

                 running = false;//signal the threads to terminate
        }
}

 //=====================================================//

class Producer extends Thread {
        public void run() {
                byte byteToStore;

                while (Synch01.running){
                        byteToStore = (byte)(Math.random()*128);
                        Synch01.queueManager.putByteInQueue(byteToStore);
                         try{
                           Thread.currentThread().sleep((int)(Math.random()*100));
                        }
                        catch(InterruptedException e){};
                }
                System.out.println("Terminating Producer run method");
        }
}
 //=====================================================//
```

```
class Consumer extends Thread {
	public void run() {
		byte dataFromQueue;

		while (Synch01.running) {
		  dataFromQueue = Synch01.queueManager.getByteFromQueue();
			try{
			  Thread.currentThread().sleep((int)(Math.random()*100));
			}
			catch(InterruptedException e){};
		 }
		System.out.println("Terminating Consumer run method");
	}
}

//=====================================================//

class QueueManager{
  Queue queue;

  QueueManager(){
	queue = new Queue();//instantiate a queue object
  }

  synchronized void putByteInQueue(byte incomingByte){
	try{
		while(queue.isFull()){
			System.out.println("Queue full, waiting");
			wait();
		}
	}
	catch (InterruptedException E){
		System.out.println("InterruptedException: " + E);
	}

	queue.enQueue(incomingByte)

	notify();
  }

  //-----------------------------------------------------//

	public synchronized byte getByteFromQueue(){
		try{
			while(queue.isEmpty()){
```

```
                                System.out.println("Queue empty, waiting");
                                wait();
                        }
                }
                catch (InterruptedException E){
                        System.out.println("InterruptedException: " + E);
                }

                byte data = queue.deQueue();

                notify();
                return data;
        }
}
//=========================================================//

//This is a standard FIFO queue class.
class Queue{
        static final int MAXQUEUE = 4;
        byte[] queue = new byte[MAXQUEUE];
        int front, rear;

        Queue(){//constructor
                front = rear = 0;
        }

        void enQueue(byte item){
                queue[rear] = item;
                rear = next(rear);
        }

        byte deQueue(){
                byte temp = queue[front];
                front = next(front);
                return temp;
        }

        boolean isEmpty(){
                return front == rear;
        }

        boolean isFull(){
                return (next(rear) == front);
        }

        int next(int index){
```

```
                return (index+1 < MAXQUEUE ? index+1 : 0);
            }
        }
```

Hasil running:

```
Queue empty, waiting
Queue full, waiting
Queue full, waiting
Queue full, waiting
Queue full, waiting
Queue full, waiting
Queue full, waiting
Queue full, waiting
Queue full, waiting
Queue empty, waiting
Queue empty, waiting
Queue empty, waiting
Queue full, waiting
Queue full, waiting
Terminating Producer run method
Terminating Consumer run method
```

Penjelasan:

Program diatas merupakan sebuah program mengenaidua buah objek yaitu producer dan consumer. Produser merupakan sebuah objek dari kelas producer yang bertindak sebagai pemasok data yang akan disimpan dalam sebuah antyrian data. Sedangkan consumer bertidak sebagai objek yang mengambil data dari dalam antrian tersebut.

Panjang antrian yang disediakan maksimal 4 antrian data. Hal ini dapat dilihat pada baris:

static final int MAXQUEUE = 4;

sehingga pada suatu saat ketika dalam antrian tersebut telah ada 4 data yang sedang mengantri dan objek producer ingin meletakkan sebuah antrian, maka akan muncul pesan bahwa antrian penuh ñQueue full, waitingò.

Sedangkan apabila objek costumer ingin mengambil data dari antrian namun antrian dalam keadaan kosong, maka akan muncul pesan bahwa antrian kosong, ñQueue empty, waitingò.

Dalam program tersebut disediakan sebuah kelas untuk menampung antrian data, yaitu class Queue. Objek dari kelas inilah yang akan digunakan sebagai penampung data dari producer.

Sedangkan proses penambahan antrian dan pengambilan data dari antrian sepenuhnya ditangani oleh kelas QueueManager. Di dalam kelas ini disediakan 2 buah method, yaitu untuk menambah antrian dan mengambil data dari antrian.

Secara umum, proses yang terjadi adalah producer meletekkan data ke dalam antrian kemuadian consumer mengambil data dari antrian tersebut.

Hasil keluaran dari dari program diatas tidak menggambarkan semua proses yang terjadi, yang ditampilkan hanya pesan ketika antrian penuh dan kosong. Untuk melihat seluruh proses yang terjadi yaitu proses penambahan antria yang berhasil, penambahan antrian yang gagal karena antrian penuh, pengambilan data yang berhasil, dan pengambilan data yang gagal karena antrian kosong, maka ditamnbahkan 4 buah baris program yaitu:
System.out.println("yang akan di qeuue " + byteToStore); pada class produser
System.out.println("yang di qeuue " + incomingByte); pada method putByteInQueue()
System.out.println("yang akan di kembalikan " + data); pada method getByteFromQueue()
System.out.println("yang berhasil diambil " + dataFromQueue); pada class consumer

Maka akan tampil keluaran sebagai berikut:
yang akan di qeuue 53
yang di qeuue 53
yang akan di kembalikan 53
yang berhasil diambil 53
Queue empty, waiting
yang akan di qeuue 12
yang di qeuue 12
yang akan di kembalikan 12
yang berhasil diambil 12
Queue empty, waiting
yang akan di qeuue 109
yang di qeuue 109
yang akan di kembalikan 109
yang berhasil diambil 109
Queue empty, waiting
yang akan di qeuue 109
yang di qeuue 109
yang akan di kembalikan 109
yang berhasil diambil 109
Queue empty, waiting
yang akan di qeuue 2
yang di qeuue 2
yang akan di kembalikan 2
yang berhasil diambil 2
yang akan di qeuue 64
yang di qeuue 64
yang akan di kembalikan 64
yang berhasil diambil 64
yang akan di qeuue 64
yang di qeuue 64
yang akan di kembalikan 64

.....//banyak dihilangkan untuk mempersingkat
.....
yang akan di qeuue 29
yang di qeuue 29
Queue full, waiting
yang akan di kembalikan 15
yang berhasil diambil 15
yang akan di kembalikan 8
yang berhasil diambil 8
yang akan di qeuue 55
yang di qeuue 55
yang akan di kembalikan 116
yang berhasil diambil 116
yang akan di qeuue 107
yang di qeuue 107
yang akan di kembalikan 29
yang berhasil diambil 29
yang akan di qeuue 79
yang di qeuue 79
yang akan di qeuue 93
yang di qeuue 93
Queue full, waiting
yang akan di kembalikan 55
yang berhasil diambil 55
yang akan di kembalikan 107
yang berhasil diambil 107
yang akan di kembalikan 79
yang berhasil diambil 79
yang akan di kembalikan 93
yang berhasil diambil 93
yang akan di qeuue 112
yang di qeuue 112
yang akan di kembalikan 112
yang berhasil diambil 112
yang akan di qeuue 1
yang di qeuue 1
yang akan di kembalikan 1
yang berhasil diambil 1
Terminating Producer run method
Terminating Consumer run method

Itulah seluruh proses yang sebenarnya terjadi.