Inter-process Communication

DR. FADI TIRKAWI

مخطط المحاضرة

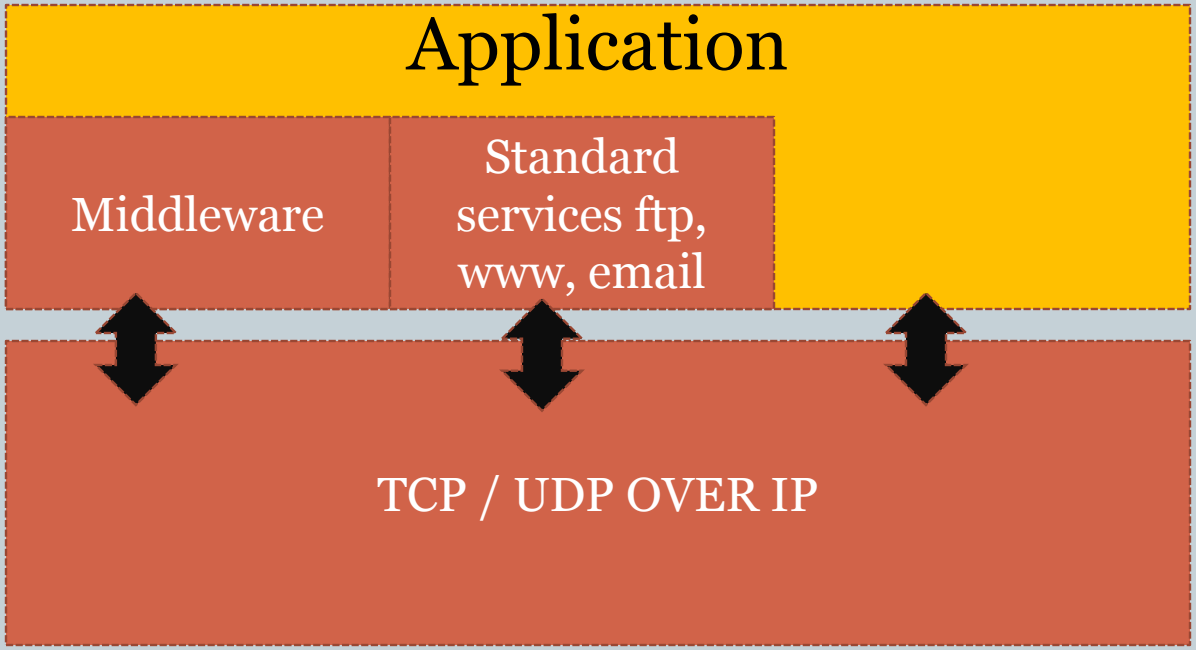
- Inter-process Communication
- الإستخدام المباشر لموصل الشبكة Network API مقابل إستخدام الطبقة الوسطية Middleware
- TCP / IP Sockets
- TCP / IP Clients and Servers in Java
- التقينات المعتمدة على هذه الموصل مثل:
- بروتوكول الطلب و الجواب Request-Reply-Protocols
- الإتصالات المتعددة
- شكل و هيئة البيانات المنقولة

Inter-Process Communication

- تطبيقات البرامج تعيش في عمليات
- العمليات هي عبارة عن كائنات لنظام التشغيل من خلالها تستطيع التطبيقات وصول بشكل أمن للمصادر. و العمليات تكون منفصلة عن بعضها.
- لكي تتصل عمليتين مع بعضهما يجب توضيع إتصال بين العمليات أو ما يدعى بـ Inter-Process Communication
- IPC على تبادل الرسائل و يكون هناك
  - عملية تقوم بإرسال رسالة و تسمى المرسل Sender
  - و عملية أخرى تسمى المستقبل Receiver

الإتصال بين العمليات المتزامن و الغير متزامن

الإتصال المتزامن	الإتصال الغير متزامن
يقوم هنا المرسل و المسقبل بقفل الإرسال و الإستقبال و هذا يعني بأن المرسل لا يمكن أن يتابع عمله إلا بعد أن تتم عملية الإرسال مع العملية المستقبلة	هنا يتم الإرسال بشكل غير مقفل non-blockingحيث يمكن أن تتابع العملية المرسلية أعمالها الأخرى بدون إنتظار وصول الرسائل للمستقبل
عندما يتم الإستقبال تنتظر العملية حتى تتم كامل عملية الإستقبال	الإستقبال يمكن أن يكون مقفل أو غير مقفل
أسهل للتنفيذ و لكن أقل فاعلية	صعب التنفيذ و معقد و لكن أكثر فاعلية

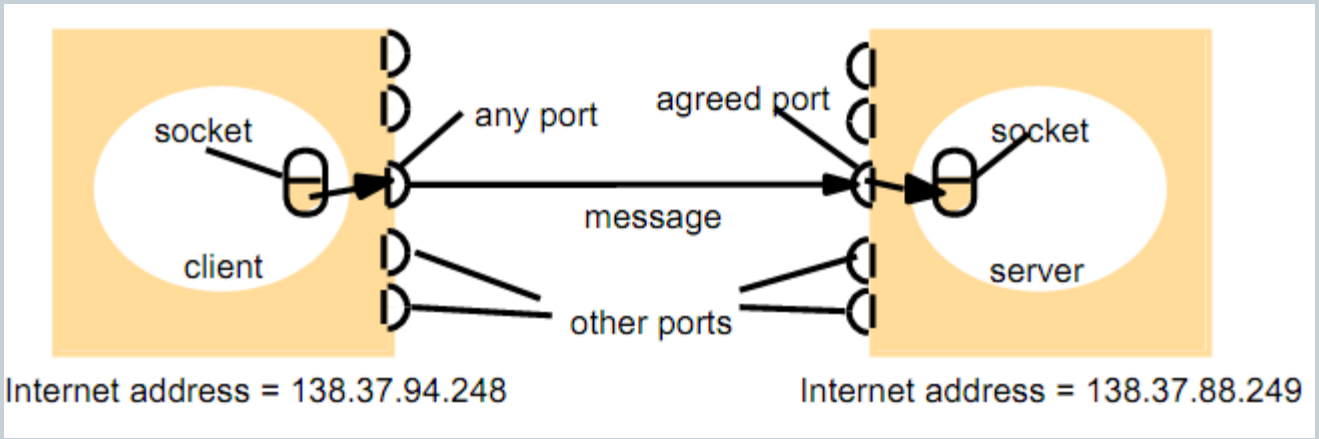

Application
Middleware
Standard services ftp, www, email
TCP / UDP OVER IP

NETWORK PROGRAMMING	MIDDLEWARE
تحكم كامل بكل معايير طبقة النقل	طريق أسهل لتطوير البرامج الموزعة
مرونة أكثر عند تتطوير التطبيقات	
يعطي أداء أعلى	يوجد حمل كبير OVERHEAD
تمثيل البيانات مشكلة هنا	تمثيل البيانات يجب أن يتم من قبل التطبيقات

TCP / IP SOCKETS

- إن استخدام واجهة البرمجة API من خلال طبقة التوصيل يسمى مقبس SOCKET.
- ان المقبس هو عبارة عن نقطة نهاية لعلاقة اتصال
- البيانات يمكن استقبالها من خلال المقبس و ارسالها من خلال المقبس
- يوجد من موصل المقبس SOCKET INTERFACE نوعين رئيسين
  - نوع مقبس مثل خط الهاتف
  - نوع مقبس مثل صندوق البريد
- المقابس يمكن ان ننفذها في كل لغات البرمجة و بالطبع JAVAاحداها.

- إن المقبس يتم تخصيصه قبل البدء بأي عملية اتصال باستخدام منفذ TCP/UDP و أيضا رقم المنفذ
- و من خلاله تستطيع تمييز الاتصال بين عملية ما و اخرى



مقابس UDP او DATAGRAM

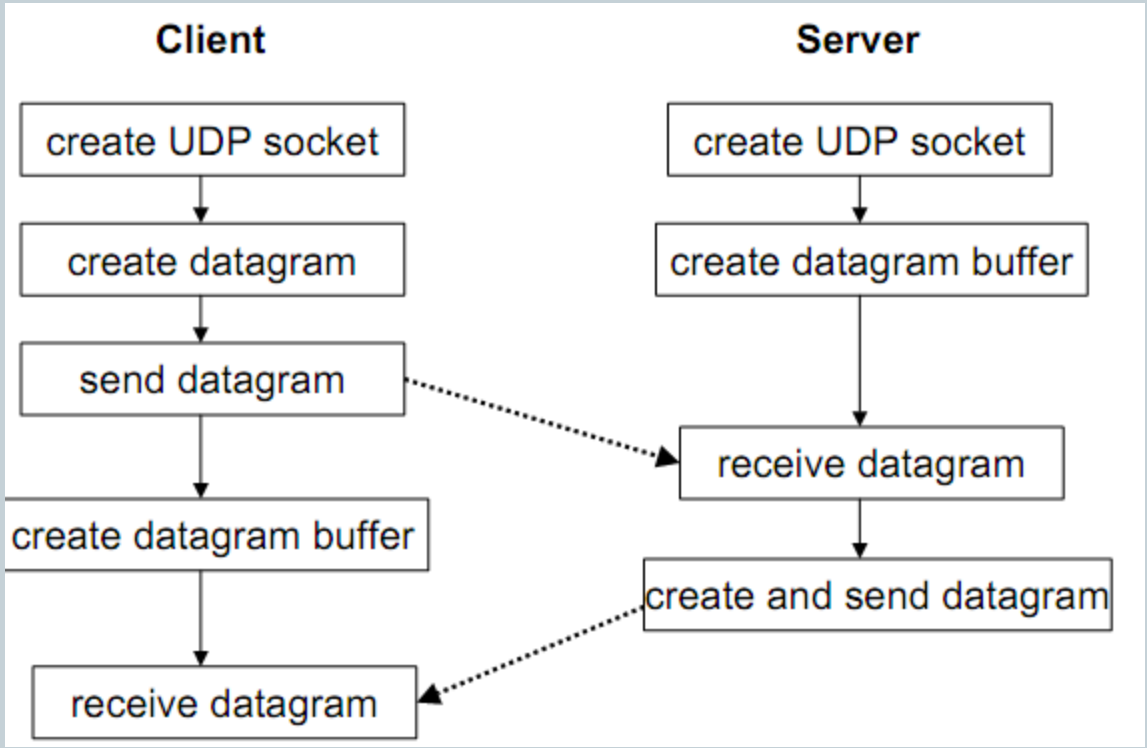
- هذا النوع من المقابس لا يتم فيه تأكيد الوصول لرسالة ما اي عند ضياع رزمة فلا يتم ارسالها
- لذلك خطأ الرزم يؤدي لضياع الرزم بدون أن يلاحظ التطبيق ذلك
- حجم الرزمة الأعظمي هو ٨ كيلو بايت و الرزم الاكبر يجب تقطيعها في التطبيقات و بالمقابل تجميعها في التطبيقات المقابلة
- ان هذا النوع من المقابس يطبق BLOCKING SEND و NON-BLOCKING SEND

JAVA API FOR UDP SOCKETS

- TWO CLASSES
  - DatagramPacket يتضمن المعلومات المرسلة
  - DatagramSocket يملك بشكل رئيسي الطرق

```
- send(DatagramPacket)
- receive(DatagramPacket)
```

Typical Structure of UDP Programs



UDP Client

```
import java.net.*;
import java.io.*;
public class UDPClient{
    public static void main(String args[]){
            try {
                    DatagramSocket aSocket = new DatagramSocket();
                    byte [] m = args[0].getBytes();
                    InetAddress aHost = InetAddress.getByName(args[1]);
                    int serverPort = 6789;
                    DatagramPacket request = new DatagramPacket(m,  args[0].length(),
                                                                    aHost, serverPort);
                    aSocket.send(request);
                    byte[] buffer = new byte[1000];
                    DatagramPacket reply = new DatagramPacket(buffer, buffer.length);

                    aSocket.receive(reply);
                    System.out.println("Reply: " + new String(reply.getData()));
                    aSocket.close();
            }catch (SocketException e){System.out.println("Socket: " + e.getMessage());
            }catch (IOException e){System.out.println("IO: " + e.getMessage());}
    }
```

UDP Server

```
import java.net.*;
import java.io.*;
public class UDPServer{
        public static void main(String args[]){
            try{
                    DatagramSocket aSocket = new DatagramSocket(6789);
                    byte[] buffer = new byte[1000];
                    while(true){
                        DatagramPacket request = new DatagramPacket(buffer, buffer.length);
                        aSocket.receive(request);
                        DatagramPacket reply = new DatagramPacket(request.getData(),
                                request.getLength(), request.getAddress(), request.getPort());
                        aSocket.send(reply);
                    }
            }catch (SocketException e){System.out.println("Socket: " + e.getMessage());
            }catch (IOException e) {System.out.println("IO: " + e.getMessage());}
    }
}
```