Intercambio de Mensajes

Hernán Melgratti

hmelgra@dc.uba.ar

Memoria compartida

- Hasta ahora vimos mecanismos de sincronización basados en variables compartidas
  - Estos mecanismos suponen que los procesos comparten memoria
  - No son adecuados para aplicaciones distribuidas
- Es la causa de muchos de los problemas de sincronización (secciones críticas)

Intercambio de mensajes

- Eliminan el supuesto de la memoria compartida
  - Un proceso envía un mensaje (*send*)
  - Otro proceso espera por un mensaje (*receive*)
- Existen dos dimensiones principales en este enfoque
  - El tipo de sincronización utilizado
  - La forma en que los procesos se identifican

Sincronización

- Envío Sincrónico
  - Enviar y esperar a que el mensaje sea recibido (un fax)
- Envío Asincrónico
  - Enviar y continuar trabajando (email, SMS)
- Rendezvous / Invocación remota
  - Enviar y esperar una respuesta (llamada telefónica)

Identificación de procesos

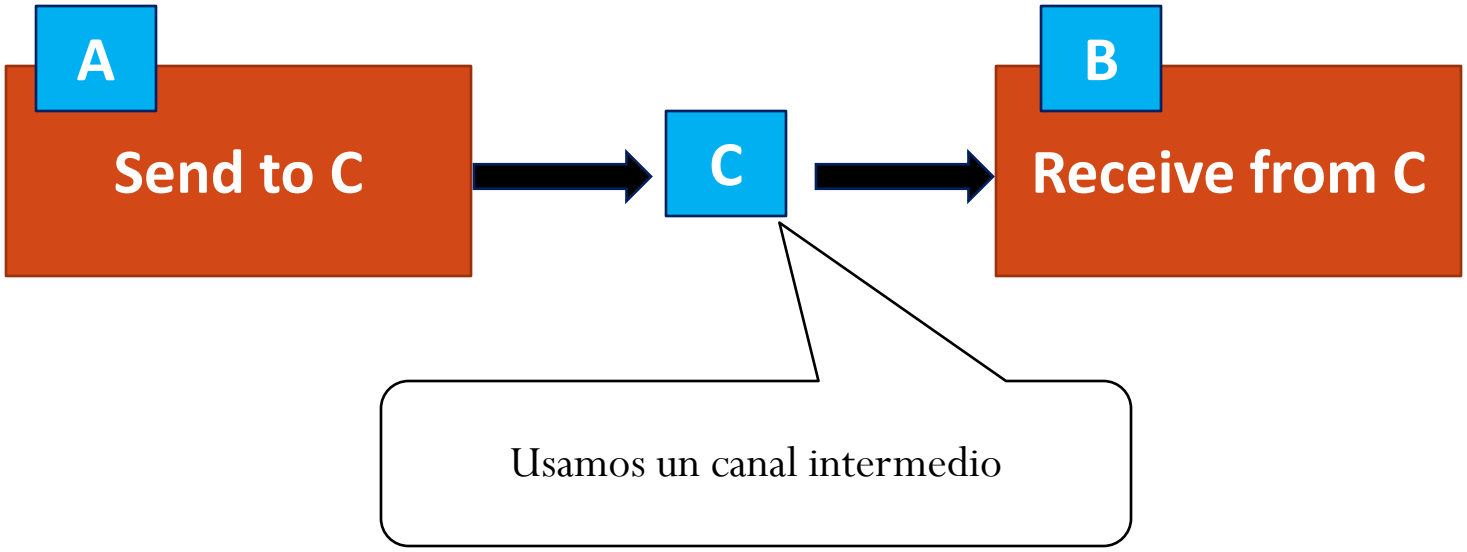
- Como hacen los procesos que envían y reciben para identificarse cuando utilizan intercambio de mensajes
- Existen tres disciplinas comunes:
  - Identificación simétrica directa

Identificación de procesos

- Como hacen los procesos que envían y reciben para identificarse cuando utilizan intercambio de mensajes
- Existen tres disciplinas comunes:
  - Identificación asimétrica directa

Identificación de procesos

- Como hacen los procesos que envían y reciben para identificarse cuando utilizan intercambio de mensajes
- Existen tres disciplinas comunes:
  - Identificación indirecta



Canales

- Los podemos pensar como buffers
- Que dimensión tienen estos buffers?
  - Si la comunicación es asincrónica: capacidad no acotada
  - Si la comunicación es sincrónica: 0 (el proceso que envía tiene que esperar a que el consumidor esté listo para leer)

Intercambio de mensajes en Java

- Sockets: intercambio asincrónico
- RMI: Puede ser visto como intercambio sincrónico

Sockets en Java

- Un sock et TCP es un punto para poder comunicar
- Es un par <dirección IP, Puerto>
  - Un puerto es un numero entero entre 1024 y 65535
- Una conexión TCP (un canal) es un par de sockets
  - Un socket client
  - Un socket servidor
  - Un socket servidor espera escuchando en un port mensajes enviados por el cliente

Lado Servidor

```
public class SocketServidor {
 public static void main(String argv[]){
    ServerSocket serverSocket;
    try {
       serverSocket = new ServerSocket(1234);
       Socket serverSoc = serverSocket.accept();
       BufferedReader serverIn = new BufferedReader(new
             InputStreamReader(serverSoc.getInputStream()));
       System.out.println(serverIn.readLine());
       serverIn.close();
       serverSoc.close();
       serverSocket.close();
       }
    catch (IOException e) {
       System.out.println("Could not listen on port: 1234");
       System.exit(-1);
       }
    }}
```

Lado Cliente

```
public class SocketClient {

 public static void main(String argv[]) {
    Socket socketcliente=null;
      try {
            socketcliente = new Socket("localhost",1234);
            PrintStream clienteOut = new
              PrintStream(socketcliente.getOutputStream());
            clienteOut.println("Hola");
            clienteOut.close();
            socketcliente.close();
      }
      catch (Exception e) {…}
    }
}
```

Múltiples servidores

```
public static void main(String argv[]) {

    try {
        ServerSocket sSoc = new ServerSocket(2001);

        while(true) {
            Socket inSoc = sSoc.accept();

            EcoThread ecoT = new EcoThread(inSoc);
            ecoT.start();}
    }
    catch (Exception e) {
    }
}
```

Múltiples servidores

```
class EcoThread extends Thread {
    Socket threadSoc;
    FibThread(Socket inSoc) {
        threadSoc = inSoc;
    }

    public void run() {
        try {
            BufferedReader serverIn = new BufferedReader(new
                InputStreamReader(serverSoc.getInputStream()));
            System.out.println(serverIn.readLine());
            …
    }
…
}
```

Ejercicio 1:

- Realizar un dos programas que comuniquen utilizando sockets:
  - ProductorSocket: Espera una conexión y genera los 20 primeros números de la serie de Fibonacci
  - ConsumidorSocket: Se conecta al servidor y muestra los 20 números que recibe del productor

Ejercicio 2:

- Modificar la solución anterior de manera tal que el cliente envía al productor la cantidad de elementos de la serie que debe generar el productor.

Ejercicio

- Modificar la solución del productor de números de Fibonacci para que acepte conexiones de muchos clientes.