Networking and Concurrency

COMP 112 2018

School of Engineering and Computer Science

Victoria University of Wellington

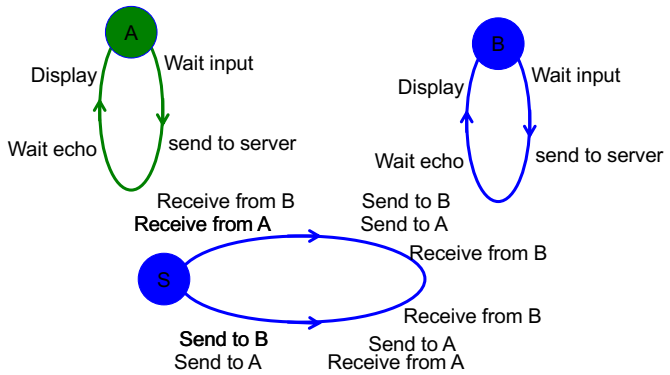
Menu

- Recall the many client problem for the Echo Server
- You need to:
  1. Understand what the problem is
  2. How the solution works
  3. Recognise a similar problem and apply the same solution

(Echo Server ----- IRC Client)

The two Client problem!

What dose the server look like if there are two clients



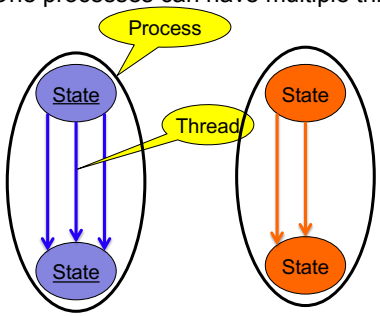
Unknown action order

Problem!

Cope with unknown execution order!

Building blocks for solution.

- Computers appear to execute two programs at once
- A process has state and a thread of execution
- Two processes have disjoint state
- One processes can have multiple threads of execution



Each thread shares the same state as all other threads of the same process

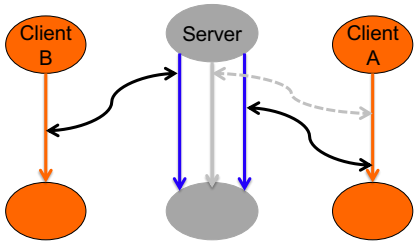
This can introduce very subtle errors!

Echo Server

- The Server may have lots of Clients and needs to process each Client with no apparent delay
- Client processing should be independent of each other
- Server canot wait for Client $\mathcal{A}$ to respond else Client $\mathcal{B}$ may have to wait utill after $\mathcal{A}$ reponds

- Echo Server has one process listening for new clients and one process listening for client comunication.
- The process listening for client comunication has one thread for each Client
- Basic Server design:
  - Listen to the port. (unknown number of clients)
  - Start a new thread for each client (unknown action order)

Concurret Echo server



- Server and Client are distinct processes
- Client connects to server (main thread)
- Server main thread runs a new thread to communicate with each Client
- Distinct clients are distinct processes
- The sever runs separate threads for each client

Concurrent Echo Server Design

1. Main Program
   - Listen to the port.
   - Loop
     - Wait for a new client to try to connect
     - create a new socket for the client connection
     - start a thread to process the client
2. Thread processing the client
   - Loop
     - listen for input from client
     - echo it back
     - quit if the message was QUIT

Problem + Solution

1. Cope with unknown execution order!

2. Use more than one thread

Echo Server: Listening for clients

```
public class EchoServer{
    public static final int PORT = 6667;  // The port the server will listen on

    public static void main(String[] args) {
        try{
            ServerSocket serverSocket  = new ServerSocket(PORT);
            System.out.println("Waiting for clients to connect...");
            while (true) {
                Socket socket = serverSocket.accept();
                System.out.println("Found client");
                EchoService echoService= new EchoService(socket);
                new Thread(echoService).start();
            }
        }catch(IOException e){System.out.println("Failed to connect" + e);}
    }
}
```

Executed when calling EchoServer

One port Many sockets

wait for a client Builds new socket

new thread

Executes constructor

Executes the run method

Echo Service: per client

```
class EchoService implements Runnable {
    private Socket socket;
    private Scanner clientIn;
    private PrintStream clientOut;

    public EchoService(Socket socket) {
        this.socket = socket;
        try {
            clientIn = new Scanner(socket.getInputStream());
            clientOut = new PrintStream(socket.getOutputStream());
        }
        catch (IOException e) {System.out.println("Connection failed." + e);}
    }
```

Constructor only sets up streams for socket

Echo Service: per client

```
    public void run() {
        while (clientIn.hasNext()) {
            String message = clientIn.nextLine();
            System.out.println("Received: "+ message);
            if (message.equals("QUIT")) { break;}
            clientOut.println("ECHO: "+ message);
            clientOut.flush();
        }
        try{ socket.close();}
        catch (IOException e) {System.out.println("Error socket close" + e);}
        System.out.println("Client disconnected.");
    }

}
```

For each client this is Executed in a separate thread

Shared memory

Note flush stream

Synchronous or Asynchronous?

- Two different echo clients are asynchronous
- Each echo client:
  - While(true)
    - get message from user
    - send message to server
    - get reply from server
    - display reply

Each client is Synchronous:
one thread sending and receiving.
Fixed cycle of interaction

- How do you want an IRC client to behave ?
- IRC client:

NO Fixed cycle of interaction

  - messages may come from the server to be displayed to the user
  - messages may come from the user to be sent to the server

⇒ client must be concurrently listening to user and server

⇒ each client must have two threads!

Each client is Asynchronous:
one thread sends user input to server
another receives data from server and displays .

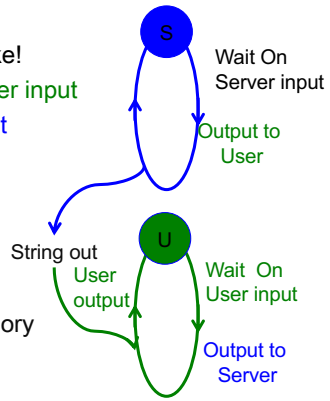
Reuse solution.

1. Cope with unknown execution order!
2. Use more than one thread
3. Similar problem similar solution

Echo server. - IRC Chat client

The Asynchronous Client problem!

- What dose async client look like!
- One thread one waiting for user input
- Another waiting for server input
- What about output?
- Assume fast so put any where
- Keep
  1. All sever IO in Server thread
  2. All user IO in User thread
- Communicate via shared memory
- Evaluate!



Asynchronous client

```
public class AsynchClient {
    private static final String SERVER = "localhost";
    private static final int PORT = 6667;

    private Socket socket;
    private Scanner input;
    private PrintStream output;

    public static void main(String[] args) { new AsynchClient(); }

    public AsynchClient(){
        try {
            socket = new Socket(SERVER, PORT);
            new Thread(new Runnable(){ public void run(){
                listenToServer();
            }}).start();
            listenToUser();
        }catch(IOException e){UI.println("IO failure "+ e);}
    }
```

Asynchronous client

```
public void listenToUser(){
    PrintStream output = new PrintStream(socket.getOutputStream());
    while (true) {
    String toSend = UI.askString(">");
    output.println(toSend);
    output.flush();
    }
}
```

```
public void listenToServer(){
    Scanner input = new Scanner(socket.getInputStream());
    while (input.hasNext()){
        String line = input.nextLine();
        UI.println("SERVER: "+ line);
    }
}
```


Threads

- What is a thread?
  - Like a separate CPU running a program (or part of a program).
  - independent of all the other threads (could be faster or slower).
- Java threads all have access to the same memory
  - Two threads accessing the same location can cause conflict and error
- Safe Programming with threads is HARD.
  - Harder to debug.
  - No problems if the different threads don't share any resources
  - You should expect some odd things to happen if the threads do share resources (eg, the same window!)

How do you get a Thread?

- The main method is called with one thread
  - anything it calls is executed in that thread:
    main → constructor
- The GUI events are executed in a separate thread called by the Java language not you, the application programmer.
  - repainting, responding to mouse, buttons etc.
- You can create a new Thread object and call run() on it.