Process communication lecture 2

Pipes

Sockets 19.9.1

IPC streams

Common to access communication through stream mechanism

just like files

Pipes

Good analogy

Data gets put into the pipe and taken out the other end

- implies buffering mechanism
- what size pipe?
- what about concurrent use - can writes interleave? etc

In UNIX it starts as a way for a process to talk to itself.

```
int  myPipe[2];

pipe(myPipe);
```

System call which creates the pipe. Returns two UNIX file descriptors - called file handles in this course.

myPipe[0] to read, myPipe[1] to write

e.g. write(myPipe[1], data, length);

Empty and full pipes

Reading processes are blocked when pipes are empty

Writing processes are blocked when pipes are full

Broken pipes

A process waiting to read from a pipe with no writer gets an EOF.

Similarly a process writing to a pipe with no reader gets signalled.

Writes are guaranteed to not be interleaved if they are smaller than the size of the pipe.

The pipe size in Linux (and most other UNIXs) is 4096.

Pipes in UNIX are commonly used to chain processes together.

The output of one process becomes the input of the next.

This can even be done from the shell.

The "|" symbol causes the stdout of the process on the left hand side to point to the pipe and the stdin of the process on the right hand side.

Limitations.

Can only be used to communicate between related processes.

The file handles are just low integers which index into the file table for this process.

The same numbers only make sense in the same process (or in one forked from it).

We mentioned that pipes work with stdin and stdout in the shell.

How are these processes related?

Named pipes.

A file which operates in the same way as an ordinary pipe.

Sometimes called FIFO files.

(Actually ordinary pipes may be stored as files too, but usually buffered in memory.)

Pipe streams in Java

PipedInputStream

PipedOutputStream

Have to connect one to the other either with a connect() method call or during constructions.

```
/ *
  PipeTester.java - shows the use of pipes and chaining of
IO classes
  written by Robert Sheehan - 30/09/97
* /
```

```
import  java.io.*;

public class PipeTester {

  public static void main(String[] args) {
    PipedOutputStream outPipe = new
PipedOutputStream();
    PipeProducer producer = new PipeProducer(outPipe);
    PipeConsumer consumer = new PipeConsumer(outPipe);
    producer.start();
    consumer.start();
  }

}

class PipeProducer extends Thread {
  PipedOutputStream outPipe;

  PipeProducer(PipedOutputStream outPipe) {
    this.outPipe = outPipe;
  }

  public void run() {
    try  {
      DataOutputStream out = new
DataOutputStream(outPipe);
      for (int i = 1; i <= 100; i++)
        out.writeInt(i);
    }
    catch (IOException e) {
      System.err.println("Producer: " + e);
      System.exit(1);
    }
```

```
  }

}

class PipeConsumer extends Thread {
  PipedInputStream inPipe;

  PipeConsumer(PipedOutputStream outPipe) {
    try  {
      inPipe = new PipedInputStream(outPipe);
    }
    catch (IOException e) {
      System.err.println("Connecting the pipe: " + e);
      System.exit(1);
    }
  }

  public void run() {
    try  {
      DataInputStream in = new DataInputStream(inPipe);
      for (;;) {
        int data = in.readInt();
        System.out.println(data);
      }
    }
    catch (IOException e) {
      System.err.println("Consumer: " + e);
      System.exit(1);
    }
  }

}
```

At the completion there is the following message:
Consumer: java.io.IOException: Pipe broken

Sockets (originally BSD UNIX)

socket - make a socket, specify the domain and protocol

UNIX domain (can be used to implement pipes)

names are filenames

Internet domain

names are IP addresses, names or numbers

plus port number

bind - associate a name with the socket

listen - now ready to get connections

accept - gets a connection and returns a new socket (used for the actual communication)

another process (for the other end of the socket)

socket - make a socket

connect - makes the connection between this socket and the named one

Port numbers

16 or 32 bit.

Bottom 1024 reserved, mail, http, ftp, telnet, echo, date etc

Only one process bound to each port.

Java simplifies things:

Just uses the Internet domain

ServerSockets and Sockets

Constructing a ServerSocket needs a port number.

then call accept() which waits for a connection

accept returns with the value of a new socket which is the socket for the communication

The other process calls Socket - needs the IP address and the port number

Sockets have Input and Output Streams associated with them.

```
/ *
  Receiver.java - demonstrates simple use of sockets
  written by Robert Sheehan - 30/09/97
* /

import  java.net.*;
import  java.io.*;

public class Receiver {

  public static void main(String[] args) {
    ServerSocket catcher;
    Socket connection;
    DataInputStream input;
    DataOutputStream output;

    try  {
```

```
      catcher = new ServerSocket(12774);  // hopefully
unused port
      connection = catcher.accept();
      input = new
DataInputStream(connection.getInputStream());
      output = new
DataOutputStream(connection.getOutputStream());
      String data = input.readUTF();
      output.writeUTF("Hi, I got: " + data);
      connection.close();
      catcher.close();
    }
    catch (IOException e) {
      System.err.println("Receiver: " + e);
    }
  }
}

/ *
  Sender.java - the other half
* /

import  java.net.*;
import  java.io.*;

public class Sender {

  public static void main(String[] args) {
    Socket connection;
    DataInputStream input;
    DataOutputStream output;

    try  {
```

```
      connection = new Socket("127.0.0.1", 12774); / /
loopback connection
      output = new
DataOutputStream(connection.getOutputStream());
      input = new
DataInputStream(connection.getInputStream());
      output.writeUTF("Hello from the sender");
      String data = input.readUTF();
      System.out.println(data);
      connection.close();
    }
    catch (IOException e) {
      System.err.println("Sender: " + e);
    }
  }
}
```

Things to explain:

loopback address 127.0.0.1

port number (12774 in this case)

look at method calls for DataInputStream, UTF - (UCS Transformation Format)

but there are calls to send bytes, ints etc.

There is another protocol as well, datagrams. Connectionless communication.

Not as reliable. Packets are routed independently. May arrive in any order.

Therefore can be faster.

Client/Server

The server must repeatedly call accept (there is a limit on the number of outstanding requests)

and start a thread to handle the communication with the client (pop-up thread)