Software Platforms

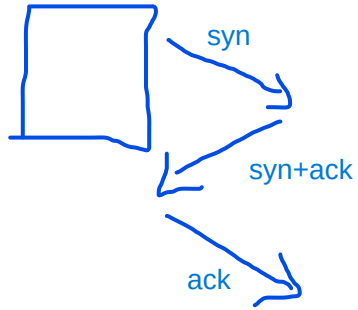
Asynchronous I/O

LM in Computer Engineering

Massimo Maresca

Anatomy of a program

- Processing (Regular CPU + Accelerators)
- I/O Mass Storage
- I/O Network



questo processo può richiedere diverso tempo

sopratutto in confronto ad operazioni normali che utilizzano la cpu

ed è per questo che si hanno piu thread di quanti cpu hai. nel mentre un f un operazione i/o molto lenta la cpu puo essere utilizzata.

Overlapping Processing and I/O: basis of concurrency

ma tanti threads sprecano risorse di memoria

Multiprogramming:

Oriented to Fast Response (Real Time Operating Systems)

Oriented to global performance optimization

ogni connessione richiede un thread. Se si utilizza la thread pool che ha un numero limitato di threads saranno in coda e non in esecuzione

ma d'altro canto non usare la thread pool puo portare ad un numero eccesivo si thread, anche perche la maggior parte del tempo un thread che esegue una connessione non fa nulla

per questo esiste l'asyncronous IO in questo modo un threads puo eseguire piu connessioni contemporaneamente, in questo modo il thread avvia una connessione e aspetta facendo altro nel frattempo

Programming with Threads vs Programming with Processes

- Concurrency at the programming level
- While processes do not make concurrency visible in application programs ad programmers just share the existing computing resources with other users…
- On the contrary Threads make concurrency visible and manageable by programmers.
- Programmers shift their attention from sequential algorithm to system level issues.

Concurrency (Processes and Threads)

- Limited vs Unlimited number of Threads.
- Too many threads
  - Make scheduling complex and inefficient.
  - Consume a large quantity of memory to keep the contexts.
- Fraction of Processing and fraction of Wait. Is it known ? Can it be estimated ?
  - Processing speed has improved a lot.
  - Latency has not.
  - Shared resource access has not.
- For example
  - Speed of light and geographical distances
  - File storage and data bases

Network Programming Interface

- Starting from SYN / SYN+ACK / SYN (three way handshaking)….
- End to end network delays
  - Transmission delays
  - Switching/Routing delays
  - Queueing delays

Multithreading is still oriented to Tasks

- A Task appears as a unit of computation, which has a start, an execution and a termination.
- During execution a Task can be associated to a status, which consists:
  - of the progress of computation (the Program Counter in terms of Assembly language, the stack, etc.), and
  - of the current context (variables and memory in general, heap, etc.).
- Network Connection, Mass Storage Access and in general latency-based operations are treated homogeneously. As their execution is overlapped with other operations (thanks to the Operating System) the number of concurrent Tasks (and Threads) may grow up and may lead to a waste of memory and processing power, as well as to long response times.

Some Numbers to have a feeling of what happens …

```
int accumulator = 0;
for (int i=0; i < iterationNumber; i++) {
        for (int j=0; j<1000; j++) {
                accumulator+=(i*j);
                if (accumulator > 1000000000)
                        accumulator = accumulator/1000;
                if (accumulator == fakeParam) {
                        System.out.println(accumulator);
                }
        }
}
```

- iterationNumber: 30000 (30M integer product and accumulation)
- time requested: 25 msec

- ping polare.cipi.unige.it from outside Unige through Fastweb: RTT ~ 40 msec

How about breaking long latency operations ?

- The Asynchronous I/O Paradigm brings thread concurrency to the programming level.
- With regular Thread Programming programmers see and manage threads as atomic units. More specifically they see Tasks and Threads and basically manage their association.
- Unfortunately, they do not take advantage of the typical task nature, i.e., of its organization as a composition of processing and waiting activities.
- The idea behind Asynchronous I/O is to break this barrier and to give programmers visibility over these aspects.

And so ? Let us start from an example

- A Task opens a socket to connect to an external server.
- Instead of just waiting for connection completion (and keeping the thread busy)… the Task releases the thread.
- So, it manages its wait status directly, at a programming level.
- Then, as soon as the connection is established (three-way handshaking), the task wakes up and proceeds.
- Question #1: Who wakes it up ? un thread qualsiasi disponibile
- Question # 2: What does “proceed” mean if the Task is not associated to a Thread ?
- A new Thread must be assigned to the execution of the task progress… until, again, a long latency operation is to be executed.

Traditional Synchronous Connection

socket di solito si usa su TCP e la loro connessione e invio dati si chiama stream

```
Socket s;
s = new Socket("127.0.0.1", 8832);
```

questa connessione potrebbe impiegare molto tempo

Asynchronous Connection

```
SocketChannel socketChannel = SocketChannel.open();
socketChannel.configureBlocking(false);
int ops = socketChannel.validOps();

Selector selector = Selector.open();

SelectionKey key = socketChannel.register(selector, ops);

InetSocketAddress myAddr = new InetSocketAddress(<address>, <port>);
boolean connected = socketChannel.connect (myAddr);
```

un modo per controllare se è ready

un selector è associato a diversi processi di i/o
quando uno finisce lo assegna ad un thread

Selector

```
while (true) {
  readyChannels = selector.select();
  if (key.isConnectable()) {
    Log ("Connectable");
    Log ("Finishing Connection...");
    boolean finished =
       socketChannel.finishConnect();
    if (finished) {
      Log ("finishConnect positive");
    } else {
      Log ("finishConnect negative");
    }
  }
```

```
if (key.isReadable()) {   c'è qualcosa nel buffer che posso leggere?
  Log ("Readable");
  ByteBuffer buf =
    ByteBuffer.allocate(1000);
  bytesRead = socketChannel.read(buf);
  Log ("byteRead = " + bytesRead);
  String s =
    new String (buf.array(), 0, bytesRead);
  Log (s);
}
```

Asynchronous I/O - NIO

```
on_connectable:
        boolean finished = socketChannel.finishConnect();

on_readable:
        ByteBuffer buf = ByteBuffer.allocate(1000);
        bytesRead = socketChannel.read(buf);
        ….
```

Asynchronous I/O – Program # 1: Asynchronous Client

- The folder includes two programs, namely MyServer.java, which is a regular server which has nothing to do with NIO, and MyNIOClient.java, which is a NIO based client aimed at showing the way NIO works.
- The server
  - places itself in wait over port 8832,
  - accepts the connection immediately,
  - waits 4 seconds,
  - sends a message (“Hi I am the server”) to the client,
  - waits other 4 seconds,
  - closes the connection.
- The client
  - creates a SocketChannel,
  - configures it as a non-blocking element,
  - creates a Selector and activates the Selector over all valid operations,
  - associates the SocketChannel to the Selector,
  - performs the asynchronous connect,
  - enters a while-true loop and waits
    - for the “connectable” operation (event), signaling connection acceptance, and later
    - for the “readable” operation (event), signaling that some data has arrived and is ready to be read.

Asynchronous I/O – Program # 2: Asynchronous Server

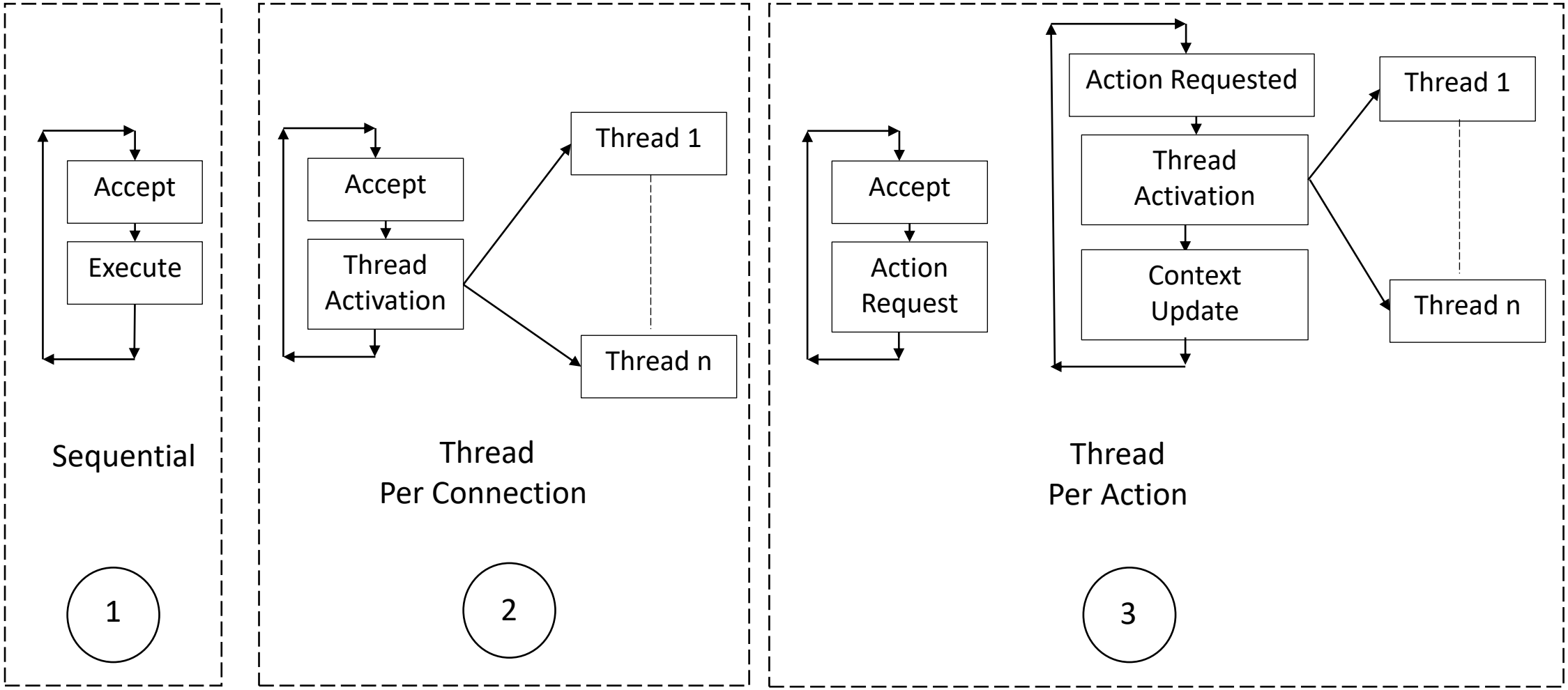
- The folder includes two programs, namely Server.java, a NIO based Server, which includes Service.java to process the messages received, and a regular Client which sends messages.

  The Server logs messages and actions.

- The server
  - Creates a SocketChannel,
  - Binds it to localhost, port 9000,
  - Registers to OP_ACCEPT operation,
  - Starts a while (forever) in which
    - It logs operations
    - Processes the Selection Keys:
      - If Acceptable it accepts the connection request and registers to OP_READ operation
      - If Readable it reads from the channel, creates a processing Thread and submit it to a pool for execution.
        The processing Thread waits 2 seconds and prints the message received.

- The client
  - Creates a Socket, connects it to localhost: 9000
  - Sends a message: *“Message from Client to Server”*

Asynchronous I/O on the Server Side: Connections, Threads and Contexts



c'è una tabella dei vari syn sent così quando ti arriva una sync+ack puoi guardare la tabella e vedere chi ha inviato la risposta. osservando il syn (che immagino sara univoco per ogni invio)

per sfruttare completamente un evento asincrono devi strutturare la tua organizzazione interna in modo adeguato

Asynchronous I/O – Program # 3: Asynchronous Client Platform

- The folder includes two programs, namely MyServer, a regular synchronous server supposed to be run in a terminal console, and MyNIOClient, a Platform able to host asynchronous clients, suggested to be activated as an Eclipse project. Let us focus on the MyNIOClient:
- The main class (Program.java)
  - activates the platform (Platform.java) in a separate thread;
  - creates and launches some Clients.
- The Client (Client.java)
  - opens a socketChannel and configures it as non-Blocking;
  - issues a connect;
  - Define the keys that are supposed to generate events and registers the socketChannel in the Platform;
  - provides methods to handle network events.
- The Platform (Platform.java)
  - run its main loop driven by external events;
  - offers two services to clients:
    - Registration: registration is requested by a client at activation;
    - Removal: removal is requested by a client application layer upon activity termination;
  - client-platform interaction takes place through two methods provided to clients;
  - actual registration/removal takes place in the platform loop under platform control.

Program # 3: Asynchronous Client Platform

How the Platform works

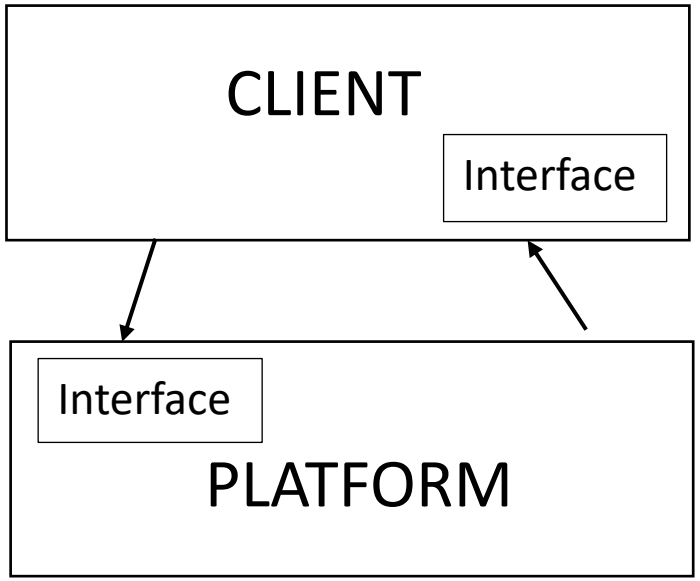
- It allows exploiting parallelism while keeping the thread number limited.
- It provides services to the application programs. In particular it offers the following services:

  public static void externalRegister(Client client)

  public static void externalSocketClose(SocketChannel socketChannel)
- It provides centralized services for I/O related operations so as to avoid thread synchronization conflicts.
- Clients enqueue requests on queues. At every iteration the platforms dequeues requests, executes them and proceeds.

Program # 3: Asynchronous Client Platform
Service requests and function provisioning

- Clients request services and provide functions called by the platform



Program # 3: Asynchronous Client Platform
Platform Organization

- An infinite loop scanning events.
- Upon the occurrence of an event the platforms takes care of the actions related to the clearance of that event and then activates the application functionalities associated to that event (onXXX interface).
- Application code may activate threads running independently with respect to the platform.
- Particular attention must be devoted to the concurrent execution of the platform thread and of the application threads.
- An additional Management Thread may monitor the platform and support management operations.

Program # 3: Asynchronous Client Platform Analysis

- Platform: an endless loop doing what follows every second (line 88):
  - Extract a Client from External Client Registration Queue
    - Do internal Client registration, i.e., attach Client to Selector
  - Extract a SocketChannel from External Closed SocketChannel Queue
    - Close the SocketChannel
  - Event Service
- Program:
  - Create Registration Queue and Closed Socket Queue
  - Create and start Platform
  - Create and activate Clients
- Client
  - Create and Register Client

- Test: Switch off client activation in the main program (Program class comment lines 47-59) and uncomment line 78 in platform (Platform class). Run the program and see that it prints a message every second. The platform is alive.

NIO Platform printout

```
3:Platform: Creating Platform
549:Client A Opening socket
555:Platform: Enqueueing registration: Client A
756:Client B Opening socket
757:Platform: Enqueueing registration: Client B
1549:Platform: Registering Client A 9
1550:Platform: Registering Client B 9
1550:Platform: Thread-0,  readyChannels = 2
1550:Platform Loop sun.nio.ch.SelectionKeyImpl@5c9b8ce2 8 Client A
1551:Platform: Acceptable = false Connectable = true Readable = false Writable = false
1551:Platform Open Client A java.nio.channels.SocketChannel[connection-pending remote=localhost/127.0.0.1:8832]
1551:Platform: Client A isConnectable
1551:Platform: Client A finishConnect positive
1552:Platform: Removing sun.nio.ch.SelectionKeyImpl@5c9b8ce2 java.util.HashMap$KeyIterator@1da686d9
1552:Platform Loop sun.nio.ch.SelectionKeyImpl@58a413a9 8 Client B
1552:Client A DoOnConnectable
1552:Platform: Acceptable = false Connectable = true Readable = false Writable = false
1553:Platform Open Client B java.nio.channels.SocketChannel[connection-pending remote=localhost/127.0.0.1:8832]
1553:Platform: Client B isConnectable
1553:Platform: Client B finishConnect positive
1553:Platform: Removing sun.nio.ch.SelectionKeyImpl@58a413a9 java.util.HashMap$KeyIterator@1da686d9
1553:Client B DoOnConnectable
```

NIO Platform printout

2558:Platform: Thread-0, readyChannels = 1
2558:Platform Loop sun.nio.ch.SelectionKeyImpl@5c9b8ce2 1 Client A
2558:Platform: Acceptable = false Connectable = false Readable = true Writable = false
2558:Platform Open Client A java.nio.channels.SocketChannel[connected local=/127.0.0.1:64743 remote=localhost/127.0.0.1:8832]
2558:Platform: Client A isReadable
2559:Platform: Client A: byteRead = 19
2560:Platform: Removing sun.nio.ch.SelectionKeyImpl@5c9b8ce2 java.util.HashMap$KeyIterator@2c8dbf04
2560:Client A DoOnReadble
2560:Client A Received: Hi, I am the Server
2760:Platform: Thread-0, readyChannels = 1
2760:Platform Loop sun.nio.ch.SelectionKeyImpl@58a413a9 1 Client B
2761:Platform: Acceptable = false Connectable = false Readable = true Writable = false
2761:Platform Open Client B java.nio.channels.SocketChannel[connected local=/127.0.0.1:64744 remote=localhost/127.0.0.1:8832]
2761:Platform: Client B isReadable
2762:Platform: Client B: byteRead = 19
2763:Client B DoOnReadble
2763:Platform: Removing sun.nio.ch.SelectionKeyImpl@58a413a9 java.util.HashMap$KeyIterator@2c71daeb
2764:Client B Received: Hi, I am the Server
4561:Platform: Thread-0, readyChannels = 1

….

Focus on Dynamic Application Loading

- Instead of having applications belong to the same project as the platform we have to resort to Dynamic Class Loading.
- The point is that we want to be able to load the applications running in the platform dynamically .
- More specifically we need to decouple platforms from applications through interfaces.
- While platform programmers focus on platform operation, management and control, on the contrary application programmers focus on applications.