SFDV3006
Concurrent Programming

Lecture 6 – Concurrent Architecture

Concurrent Architectures

- Software architectures identify software components and their interaction
- Architectures are process structures together with the way processes interact
- Aim to ignore the details concerned with the application
- Study structures that can be used in many different situations and applications

Concurrent Architectures

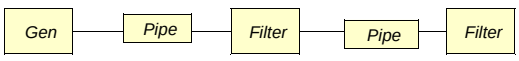
- We identify a particular architectural style
- Architectural styles are re-occurring patterns of components and connectors
- Common styles
  - Fork join – which we have in the beginning labs
  - Filter pipelines
  - Supervisor workers
  - Announcer listener
- Each of these commonly occur in concurrent and distributed systems

Filter Pipelines

- Filters receive input value stream and transform them into output value stream
- Filters are connected by pipelines
  - Redirect output of one filter to input of the next
  - May buffer values to de-couple processes from each other
- Example (from UNIX)
  - cat 1.txt 2.txt 3.txt | sort | more
  - A UNIX pipe is basically a buffer that buffers bytes of data output by one filter until they are input to the next filter.

Example: Primes Sieve

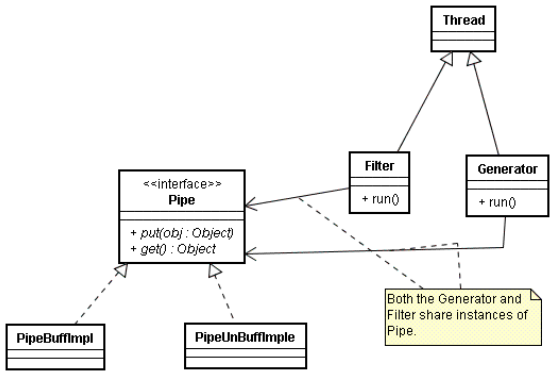
- Aim: compute primes between 2..N
- Idea:
  - Generate a stream of numbers 2..N
  - Create one filter for each number between 2 and N that filter all the numbers that are multiples and only outputs others
- Leads to a filter pipeline



Primes Sieve Model

- The diagram on the previous slide is a structure diagram that describes the high-level structure of the program
- Interaction between GEN and a FILTER is through a PIPE process.
- In software architecture, pipes are called connectors. Connectors encapsulate the interaction between components of an architecture

Java Primes Sieve Implementation



Pipe buffered implementation

```
//Pipe.java
interface Pipe {
    public boolean put(Object obj);
    public Object get();
}

//PipeBufferedImpl.java
import java.util.*;

public class PipeBuffImpl implements Pipe{

    private List buffer = new ArrayList();

    public synchronized boolean put(Object obj){
        return buffer.add(obj);
    }

    public synchronized Object get() throws InterruptedException{
        while(buffer.isEmpty()) wait(); //pipe empty - wait
        Object obj = buffer.remove(0);
        return obj;
    }
}
```

Is this similar to something we covered before?

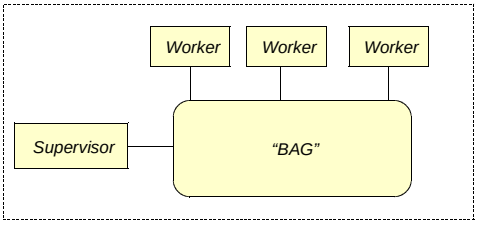
Applications for Pipelines

- Java to HTML converter: converts a Java program to HTML. We could have a filter which adds line numbers to the Java program and another filter which converts the Java source to HTML
- Text classification for Web pages:
  - Problem: HTML is used for document formatting
  - Remove all HTML tags from Web Page (HTML filter)
  - Remove all stop words from the cleaned web page (stop word filter)
  - The resulting text can now be submitted to some text classification software such as SVMs etc

Supervisor / Worker

- Supervisor-worker is a concurrent architecture that can be used to exploit parallel execution on multiple processors
- Can be applied when a computational problem can be split into a number of independent sub-problems. These independent sub-problems are called tasks.
- Supervisor creates tasks and puts them into a bag.
- A bag is connector which lets supervisor and workers interact
- Workers pick tasks from the bag and perform them
- Workers can themselves be supervisors.

Supervisor / Worker architecture



- *The supervisor collects results from the bag and determines when the computation has finished*
- *Each worker repetitively takes tasks from the bag, computes the result for that task and places the result in the bag*
- *This process is repeated until the supervisor signals that the computation has finished*

Linda Tuple Spaces

- Linda is a set of primitive operations used to access a data structure called a tuple space.
- A tuple space is shared associative memory consisting of a collection of tagged records called tuples
- Each data tuple is of the form:

  `("tag", value1,value2…., value n)`
- Tuples are exchanged in tuple spaces using associative memory
- A tuple space can be used for implementing bag connectors

Linda tuple spaces - operations

- There are three basic operations for managing data tuples: **`out`**, **`in`** and **`rd`**
- A process deposits a tuple in a tuple space using an **`out`** operation

  `out("tag", expr1, expr2…, expr n)`
- A process removes a tuple from the tuple space using an **`in`** operation

  `in("tag", field1, field2,…,fieldn)`
- The arguments to an **`in`** are called a template. The process executing **`in`** blocks until the tuple space contains a tuple that matches the template and them removes it.
- A template matches a tuple in the following situations:
  - the tags are identical
  - the template and the tuple have the same number of operations
  - The variables in the template have the same type as the corresponding values in the tuple

Linda Tuple space operations contd

- The third basic operation is **`rd`**, which is the same as **`in`** except that the tuple matching the template is not removed from the tuple space.

  `rd("tag", field1, field2,…,fieldn)`
- **`rd`** is used to examine the contents of a tuple space without modifying it.
- Linda also provides non-blocking versions of **`in`** and **`rd`** called **`inp`** and **`rdp`** which return true if a matching tuple is found and return false otherwise.

Tuple Space Implementation

```
public interface TupleSpace {
    //deposits data in tuple space
    public void out(String tag, Object data);

    //extracts object with tag from tuple space
    public Object in(String tag) throws InterruptedException;

    //reads object with tag from tuple space
    public Object rd(String tag) throws InterruptedException;

    //extracts object if available else return null – non-blocking version on in()
    public Object inp(String tag);

    //read object if available else return null – non-blocking version of rd()
    public Object rdp(String tag);
}
```

See the last slide for a simple implementation class for the *TupleSpace* interface

Tuple spaces are distributed over many processors connected by a network – this is the case with ***JavaSpaces*** (from Sun), ***TSpaces*** (from IBM) and other implementations

Supervisor-Worker algorithm

- Supervisor:

  forall tasks do out("task",…) end

  forall results: in("result",…) end

  out("stop")
- Worker:

  while not rdp("stop") do

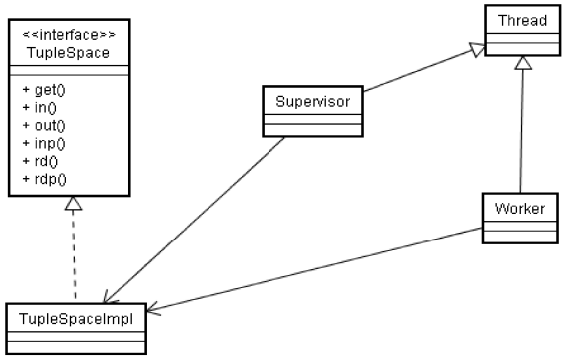
    in("task",…)

    compute result

    out("result",…)

  end

Supervisor-Worker Design



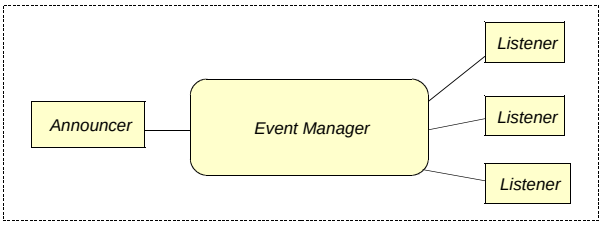
Speedup and efficiency

- Supervisor-Worker architecture is used for speedup and better efficiency
- The speedup of a parallel algorithm is defined to be the time that a sequential program takes to compute a given problem divided by the time a parallel program takes to compute the problem on N processors.
- Efficiency is speedup divided by the number of processors N.
- If a problem takes 12 seconds to compute sequentially and 4 seconds to compute on 6 processor then the speedup is 3 and efficiency is 0.5 or 50%

Announcer-Listener

- An example of event-based architecture
- The announcer announces that some event has occurred and notifies it to all the listeners that are interested in this event.
- The communication is between one announcer and zero or more listeners.
- Announcer does not know listeners
- Listeners do not know announcer
- Communication is managed by a connector called “event manager”
- Listener process indicate their interest in a particular event by registering for that event.

Announcer-Listener architecture



- *Listener can choose to listen only to a subset of events announced by registering a “pattern” with the event manager*
- *Only events that match the pattern are forwarded to the listener*
- *Listeners can also announce events to another set of listeners*
- *Listeners do not have to be active processes they can be objects.*

Application examples

- User Interface Frameworks
  - AWT listener are ordinary objects
  - Events are mouse clicks, button presses
  - Event cause operations to be invoked in listeners
- CORBA event service
  - Event producers are Announcers
  - Event channels are Event Managers
  - Event consumers are Listeners
  - Used, for example in distributed stock tickers