Assignment 2 — Reactor pattern

T-106.420 Concurrent programming

Jan Lönnberg

Department of Computer Science
Helsinki University of Technology

15th November 2005

Introduction

The Story

- As a Hangman fan, you want to create a networked version of the game.
- Multiple players co-operate in guessing a word.
- Also being a fan of design patterns, you decide to use the Reactor design pattern for the game server.

Goal

- Implement the Reactor pattern.
- Implement a simple client-server implementation of multiplayer Hangman.

Task

You get:

- Reactor API declarations.

Your task is to:

- Implement the Reactor pattern.
- Write a multiplayer Hangman based on the Reactor pattern.

Task

Doing the assignment

- Group size: 1–2.
- Submission before 2005-12-16 03:00.
- Submit a tar.gz archive containing:
  - Reactor implementation.
  - Hangman implementation.
  - Report in PDF form.
- Full instructions will be on the course home page.
- Grading: fail/pass/pass with honours.

Reactor API

Java interface

- `public class Dispatcher`
  - Implements the Dispatcher part of the Reactor pattern.
  - `public Dispatcher()`
    - Create a new `Dispatcher` with no `EventHandler`s.
  - `public void handleEvents() throws InterruptedException`
    - Waits for an event and processes it (usually called repeatedly in a loop).
  - `public void addHandler(EventHandler h)`
    - Start dispatching incoming events for `h`.
  - `public void removeHandler(EventHandler h)`
    - Stop dispatching incoming events for `h`.

Reactor API

Java interface

- `public interface EventHandler`
  - Handles events from its `Handle`.
  - `public Handle getHandle()`
    - Get the `Handle` from which the `EventHandler` receives events.
  - `public void handleEvent(Object s)`
    - Handle an incoming message represented by an object `s`, as received from the `Handle`.

Reactor API

Java interface

- `public interface Handle`
  - Represents one end of a (possibly bidirectional) communications channel.
  - `public void write(Object s)`
    - Send object `s` as a message on the channel.
  - `public Object read()`
    - Wait for a message from the channel and return it (or an object representing an error condition).
  - `public void close()`
    - Close the channel.

Implementation

Synchronisation

- Use the basic Java synchronisation primitives (those built into the language and `java.lang`, not `java.util.concurrent`).
- Inefficient solutions, such as polling and busy-waiting, will be rejected.
- Use of unsafe methods such as `java.lang.Thread.destroy()` will also lead to rejection.

Hangman

Processes in a game

- One server based on the Reactor pattern
- 0 or more clients (players).
- The server and clients run in separate virtual (possibly physical) machines.

Startup

- Server parameters:
  - Word to be guessed.
  - Number of incorrect guesses needed to lose.
- Client parameters:
  - Network name of server.
  - Name of player.

Hangman

Gameplay

- The clients guess one letter at a time.
- Each time a client guesses a letter, the server sends each client a message describing the state of the guessing so far.
- Players share guesses. Execution is completed when the amount of remaining attempts reaches zero.
- After the message describing the last guess is sent and processed, the server and all clients should be terminated.
- The Hangman client must use standard input and output for its user interface.
- Clients may connect and disconnect at any time.

Conclusion

Conclusion

- Assignment is intended to help learn monitors and condition variables, Java threads and client-server programming (using the Reactor pattern).
- Assignment description will be linked from course home page.
- Technical questions and clarification requests to the newsgroup (`opinnot.tik.rinnakkaisohjelmointi`).
- Clarifications (if any) will be posted to the newsgroup.
- Questions not intended for the public to `jlonnber@cs.hut.fi`.