Thread Pool Pattern

Thread Pooling

Group of threads created to perform a number of tasks

A thread
  Reads a task from a queue
  Performs the task
  Repeat

See http://en.wikipedia.org/wiki/Thread_pool_pattern
Server Options

Iterative Server - server handles one client at a time

Concurrent Server with Thread creation
  Create new thread for each client

Concurrent Server with Thread Pool

Concurrent Server with expandable Thread Pool

Single thread handles multiple clients concurrently
Iterative Server - When to use

**Iterative Server**

while (true)
{
    Socket client = serverSocket.accept();
    Sequential code to handle request
}

When usable

TP = Time to process a request

A = arrival time between two consecutive requests

Then we need TP << A
Concurrent Server with Thread creation

Basic Concurrent Server

while (true)
{
    Socket client = serverSocket.accept();
    new HandleClientThread(client).start();
}

When usable

Let TC = time to create a thread

Let A = arrival time between two consecutive requests

We need TC << A

Often this is good enough
## Time to Create thread

<table>
<thead>
<tr>
<th>Threads Created</th>
<th>Time - Java</th>
<th>Time - Smalltalk</th>
</tr>
</thead>
<tbody>
<tr>
<td>10,000</td>
<td>1,368</td>
<td>58</td>
</tr>
<tr>
<td>20,000</td>
<td>1,549</td>
<td>99</td>
</tr>
<tr>
<td>80,000</td>
<td>6,783</td>
<td>197</td>
</tr>
<tr>
<td>160,000</td>
<td>13,427</td>
<td>485</td>
</tr>
</tbody>
</table>

Time in milliseconds

- Run on 2.13 GHz
- Intel Core 2 Duo
- 4GB memory

Thursday, November 15, 12
Problem with Threads

Thread consume resources
  Memory
  CPU cycles

A program has a limit of
  Threads it can productively support
  Sockets it can have open

We need to insure we don’t create too many threads
Concurrent Server with Thread Pool

Create N worker threads
while (true)
{
    Socket client = serverSocket.accept();
    Use an existing worker thread to handle request
}

When usable

TP = Time to process a request
A = arrival time between two consecutive requests
N = Thread Pool size

Then we need TP << A * N
Concurrent Server - expandable Thread Pool

Create N worker threads
while (true)
{
    Socket client = serverSocket.accept();
    if worker thread is idle
        Use an existing worker thread to handle request
    else
        create new worker thread to handle the request
}

When usable

Number of requests we can handle in a unit of time

\[
\frac{TP}{N} + \frac{1}{TC}
\]

where N is not constant
Thread Pool Issues

How many threads?

When to create more threads?

When to destroy some threads?

What happens when threads stop working
Java ThreadPool Classes

java.util.concurrent.ExecutorService
  Simple interface
  Uses 3 common configurations for the pool

java.util.concurrent.ThreadPoolExecutor
  Used by ExecutorService
  Configurable
ExecutorService Example

class Server extends Thread {
    private final ServerSocket serverSocket;
    private final ExecutorService pool;

    public Server(int port)
        throws IOException {
        serverSocket = new ServerSocket(port);
        pool = Executors.newCachedThreadPool();
    }

    public void run() {
        try {
            for (;;) {
                pool.execute(new Handler(serverSocket.accept()));
            }
        } catch (IOException ex) {
            pool.shutdown();
        }
    }
}

class Handler implements Runnable {
    private final Socket socket;
    Handler(Socket socket) {
        this.socket = socket;
    }

    public void run() {
        // process request
    }
}