Concurrent Systems

Last updated: January 22^{nd} 2016, at 10.23am

Contents

1 Lecture Plan

Week	Lec.	PM	AM	Topic
13	22/1	22/1	29/1	Introduction to Concurrent Systems
14	29/1	29/1	5/2	Dekker's Algorithm
15	5/2	5/2	12/2	Semaphores
16	— Friday before guidance week —			
17	19/2	19/2	26/2	Monitors
18	26/2	26/2	4/3	Quantum Computing
19	4/3	4/3	11/3	Quantum Computing
20	11/3		18/3	Quantum Computing
21	18/3	18/3		Correctness
— Easter —				
	12/4		15/4	
22	15/4	15/4	22/4	Complexity
23	22/4	22/4	29/4	Computability
24	29/4			— Recap —

Notes

Start of term 22/1 no AM practicals

Guidance week 12/2 AM practicals *only*

Guidance week 19/2 lecture and PM practicals

Easter 18/3 *all* practicals as usual, lecture for PM practical students

Easter 12/4 lecture for AM practical students in W1/62

Easter 15/4 lecture and all practicals as usual

End of term 29/4 no PM turorials

2 What are concurrent systems?

> *Concurrent programming is the name given to programming notations and techniques for expressing* **potential** *parallelism and solving the resulting synchronisation and communication problems.*

Ben-Ari
Principles of Concurrent Programming
Prentice-Hall
1982

3 Why programme concurrent systems?

- Because they are efficient.

  Deterministic polynomial *vs.* nondeterministic polynomial

- Because they simplify programming.

  GUIs

- Because you have to.

  Operating systems.

3.1 Efficiency

3.1.1 Sequential merge sort

Algorithm

```
public void mergeSort() {
   int half; Sort left,right;
   if (size > 1) {
      half = size/2;
      left = new Sort(list,0,half-1);
      right = new Sort(list,half,size-1);
      left.mergeSort(); right.mergeSort();
      merge(left,right);
   }
}
```

Complexity

- Assume `merge` takes N "time units" t.
- How many merges?

$$n\left\{\begin{array}{cccc|cccc}
1 \text{ merge} & \text{each } N & = & 2^n & 1 & \times & N & = Nt \\
2 \text{ merges} & \text{each } \frac{N}{2} & = & 2^{n-1} & 2 & \times & \frac{N}{2} & = Nt \\
 & & \vdots & & & & & \\
2^{n-1} \text{ merges} & \text{each } \frac{N}{2^{n-1}} & = & 2 & 2^{n-1} & \times & \frac{N}{2^{n-1}} & = Nt \\
2^n \text{ merges} & \text{each } \frac{N}{2^n} & = & 1 & 2^n & \times & \frac{N}{2^n} & = Nt
\end{array}\right.$$

  So $n \times Nt$. What is n? $2^n = N \Rightarrow n = \log N$.

- So (sequential) mergesort $tN \log N$.

3.1.2 Parallel merge sort

Algorithm

```
public void mergeSort() throws InterruptedException {
   int half; Sort left,right; // Note: Sort extends Thread
   if (size > 1) {
      half = size/2;
      left = new Sort(list,0,half-1);
      right = new Sort(list,half,size-1);
      left.start(); right.start();
      left.join();  right.join();
      merge(left,right);
   }
}
```

Complexity

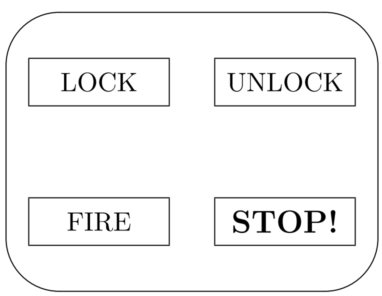
- Merges at each level can be executed in parallel

$$\begin{array}{rlcl|ccc}
1 \text{ merge} & \text{each } N & = & 2^n & 1 \times N & = & N\ t \\
2 \text{ merges} & \text{each } \frac{N}{2} & = & 2^{n-1} & 1 \times \frac{N}{2} & = & \frac{N}{2}\ t \\
 & & \vdots & & & & \\
2^{n-1} \text{ merges} & \text{each } \frac{N}{2^{n-1}} & = & 2 & 1 \times \frac{N}{2^{n-1}} & = & \frac{N}{2^{n-1}} t \\
2^n \text{ merges} & \text{each } \frac{N}{2^n} & = & 1 & 1 \times \frac{N}{2^n} & = & \frac{N}{2^n}\ t
\end{array}$$

  So $\Sigma_{i=0}^{n} \frac{N}{2^i} = N + \frac{N}{2} + \frac{N}{4} + \cdots + 1 = 2N$

- So (parallel) mergesort: $2Nt$.

3.2 Simplification



Sequential	Parallel
`while (true) {`	`while (true) {`
`LOCK.listenTo();`	`LOCK.listenTo()` $\|$
`UNLOCK.listenTo();`	`UNLOCK.listenTo()` $\|$
`FIRE.listenTo();`	`FIRE.listenTo()` $\|$
`STOP.listenTo();`	`STOP.listenTo()` $\|$
`}`	`}`

The sequential system imposes an ordering on the buttons. The code of the `listenTo`s may become complex in order to ensure that, for example, the **STOP!** button *always* prevents any of the other buttons from working. The parallel version may also require some complex code (see later weeks on e.g. critical sections) but is conceptually clearer.

3.3 Necessity

Operating Systems

- I/O devices
- Interrupts
- Multi-tasking
- Networks

4 Aspects of concurrent systems

Note: A concurrent system is not necessarily truly parallel — timeslicing, interleaving.

4.1 Necessary tools

- Communication
- Synchronisation

4.2 Properties

- Complexity
- Correctness
- Granularity

5 This semester's course

5.1 Concurrent systems

5.1.1 Properties of concurrent systems

- Critical sections
- Mutual exclusion
- Deadlock
- Starvation
- Liveness
- Loosely connected

5.1.2 Tools for concurrent systems

- Shared variables
- Semaphores
- Monitors

5.2 Quantum computing

- Quantum systems
- Circuits as matrices and vectors
- Quantum circuits
- Quantum algorithms

5.3 Theoretical aspects

- Correctness
- Complexity
- Computability

5.4 Outcomes

1. Discuss the classification of algorithms accoroding to efficiency and complexity
2. Prove code correct
3. Demonstrate a knowledge of the characteristics of a range of concurrency paradigms
4. Explain the difference between classical and quantum computing
5. Use a standard notation to analyse the efficiency and complexity of algorithms

5.5 Books

Jeff Magee & Jeff Kramer
Concurrency: State Models & Java Programs
Wiley, 2006

Noson S. Yanofsky & Mirco A. Mannucci
Quantum Computing for Computer Scientists
Cambridge University Press, 2008

Jeffrey J. McConnell
Analysis of Algorithms: an Active Learning Approach
Jones & Bartlett, 2008

6 Parallel processes in Java

6.1 Defining process classes

A parallel process is an instance of a `Thread` — a `Thread` runs a `Runnable`.

- Either implement the `Runnable` class

  ```
  class Process implements Runnable {...}
  ```

- or extend the `Thread` class

  ```
  class Process extends Thread {...}
  ```

6.2 Defining process behaviour

```
public void run() {
   ...
}
```

6.3 Creating a process

- From a subclass of `Thread`

```
Process process = new Process();
```

- From an implementation of `Runnable`

```
Thread thread = new Thread(new MyRunnable());
```

Note: this does not start the thread running. Note also that named threads can be defined:

```
Thread process = new Process(threadGroup,"My process");
```

or

```
Thread thread = new Thread(threadGroup,new MyRunnable(),"My process");
```

6.4 Starting a thread

```
myThread.start();
```

6.5 Waiting for a thread to stop

```
try {
   myThread.join();
} catch (InterruptedException e) {} ;
```

6.6 Sharing data between processes

- a non-static variable is unique to the instance

```
int belongsToPooh;
```

- a **static** variable is shared by all instances of the class

  ```
  static int botherItsPigletsToo;
  ```

6.7 Some useful methods

6.7.1 Access

- `someThread.checkAccess()`
  Is the currently running thread allowed to modify `someThread`?
- `someThread.getId()` (returns a **long**)
- `someThread.getName()` (returns a `String`)

6.7.2 Control

- `join`:
  - `someThread.join()`
    Wait for `someThread` to die
  - `someThread.join(millis)`
    Wait at most `millis` ms for `someThread` to die (millis is **long**)
- **static void** `sleep(millis)`
  Currently executing thread sleeps for `millis` ms.
- **static void** `yield()`
  Currently executing thread temporarily allows another thread to execute.

6.7.3 Priorities

- **static void** `setPriority(`**int** `newPriority)`
- **int** `getPriority()`
- `MAX_PRIORITY`, `MIN_PRIORITY`, `NORM_PRIORITY`

End of intro. to concurrent systems lecture