Principles of Concurrent Programming

Oscar Palm

Jan 2023

Contents

Chapter no. 1

Introduction to Concurrent Programming

1.1 Practical Information

1.1.1 Labs

Create a zoom meeting without a password
Put a request on waglys
The demos/labs on fridays at 08:00 is online, all others are physical.

There's three main labs:

1. Trainspotting (Java)
2. CChat (Erlang)
3. A-mazed (Java)

1.1.2 Course overview

Three parts:

1. Classic shared-memory
2. Message-parsing
3. Parallellizing computation

1.1.3 Examination

It's an open-book exam, meaning you can bring:
up to 2 textbooks,
maximum of 4 two-sided A4 sheets (printed or handwritten),
and an English dictionary.

1.2 Technical Information

1.2.1 Motivation

Imagine a sequential counter, if you then count twice you'll always end up with 2. Now imagine the same counter implemented concurrently and we run the count function in separate threads, we now have no idea which will be executed first. More importantly we now have no way of checking that they don't run simultaneously, in a worst case they may even read the value before the other thread have saved their value, meaning they overwrite each others answers.

1.2.2 Amdahl's Law

If we have n processors that can run in parallel, how much speedup can we achieve?

$$\text{speedup} = \frac{\text{sequential execution time}}{\text{parallel execution time}}$$

$$\text{Maximum speedup} = \frac{1}{(1-p) + \frac{p}{n}}$$

This means that adding more processors running in parallel benefits the program less and less, the more you add.

1.2.3 Terminology

Processes

A process is an independent unit of execution.

- Identifier
- Program counter
- Memory space

Process states

The scheduler is the system unit in charge of setting process states:

- Ready

  Ready to be executed
- Blocked

  Waiting for event before execution
- Running

  Currently running on the CPU

Threads

A lightweight process, independent unit of execution in the same program space

- Identifier
- Program counter
- Memory

  local memory, each thread has its own

  global memory, shared throughout all threads

Shared memory vs. message sharing

Shared memory communicate by writing to the shared memory space while
Distributed memory communicate by sending messages between each others.

1.2.4 Java threads

Starts with the start() method, which in turn calls the method run(). If you need to wait for a thread to finish its course, run join().

There's two different ways of generating threads in Java, either through extending the class with Thread, or by implementing Runnable. More often than not you implement Runnable because you can have an unlimited amount of interfaces but only one superclass.