Introduction to Concurrency

Why do we need/have concurrency?

- Physics, engineering, electronics, biology, …
- Sequential processing is suggested by most core computer architectures
- To employ / design concurrent parts of computer architectures

- Parallel Machines & (operating systems, compilers, databases, …)
- … to enhance performance
- … to be executed on one CPU

- MIMD
- SIMD
- General network architectures

Why would a computer scientist consider concurrency?

1. What appears sequential on a higher abstraction level, might be concurrent at a lower level. But we need to add a number of further crucial concepts

- Multiprocessor systems
- Parallel Machines & distributed operating systems
- SIMD
- General network architectures

Does concurrency lead to chaos?

Concurrency often leads to the following issues / problems:

- non-determinism: observable
- non-determinism: observable
- non-determinism: observable
- non-determinism: observable
- non-determinism: observable
- non-determinism: observable
- non-determinism: observable
- non-determinism: observable

In order to model and control such a system, its design needs to be considered.

Meaningful employment of concurrent systems:

- non-reproducible
- non-reproducible
- non-reproducible
- non-reproducible
- non-reproducible
- non-reproducible

...but we need to add a number of further crucial concepts.

References for this chapter

- Literally 'concurrent' means:
  - literally concurrent means:
  - adj.: running together in space, as parallel lines; going on side by side, as proceedings occurring together; occurring togeth-er, conjoint, associated (Oxford English Dictionary)

- Technically 'concurrent' is usually defined negatively as:
  - no observer who can identify two events as being in strict temporal sequence (i.e. one event has fully terminated before the other one started) then these two events are considered concurrent.
The concurrent programming abstraction

- No interaction between concurrent system parts means that we can analyze them individually as pure sequential programs and/or control.
- Interaction occurs on lower level.
- Communication (implicit interaction).
- Multiple concurrent execution units compete for shared resource.
- Communication (explicit interaction).

Models and Terminology

Introduction to Concurrency

Thread

- Control-flows: data in data stream.
- Threads controlled by kernel-level scheduling.
- Outside the OS: user-level scheduling.
- Threads can be terminated through normal exit.
- Synchronous Multiprocessing (SMP)

Process

- Processes can share memory, and the specific relationship of threads differs in different operating systems and architectures.
- Threads can be terminated in a group of processes, which share resources in a pre-existence in SMP.
- Each thread can be terminated after the kernel process.
- Thread and thread communication can be protected.
- Scheduling of threads depends on the architectural implementation.
- e.g., POSIX control flow, which has no knowledge about all.
- e.g., kernel-level control flow, which is handled with some extra logic.

Processes - Threads

Liveness properties

- Partial correctness: P (Q) = Process (Q) = Q (Q).
- Total correctness: P (Q) = Process (Q) = Q (Q).

Safety properties

- P (Q): Process (Q) = Q (Q).
- P (Q) = Process (Q) = Q (Q).

Examples

- Mutual exclusion (no race conditions).
- Absorptive deadlock.
- Specified responsiveness or free capabilities (typical in real-time / embedded systems or server applications).

Process Control Blocks

- Process ID
- Process state
- Process ready
- Process running
- Process terminated
- Process signals
- Process attributes
- Process struct
- Process control blocks (PCBs)

Correctness of concurrent non-real-time systems

- Partial correctness:
- Total correctness:

Logica correctness:

- Time-line or Sequence?

Consider the sequence of interaction points only:

Interalleavings (interleavings)

In non-real-time systems, e.g. 'fairness'

Partial correctness:

- Total correctness:

Slight changes in external triggers may (and usually do) result in completely different schedules (interleavings).

Correctness vs. testing in concurrent systems:

- Slight changes in external triggers may (and usually do) result in completely different schedules (interleavings).

- Consider the sequence of interaction points only:

Safety properties:

- P (Q) = Process (Q) = Q (Q).
- P (Q): Process (Q) = Q (Q).

Examples

- Mutual exclusion (no race conditions).
- Absorptive deadlock.
- Specified responsiveness or free capabilities (typical in real-time / embedded systems or server applications).

Process Control Blocks

- Process ID
- Process state
- Process ready
- Process running
- Process terminated
- Process signals
- Process attributes
- Process struct
- Process control blocks (PCBs)
Introduction to Concurrency

Process states

- Created: The task is ready to run, but
- not yet admitted or dispatched for running
- Ready: The task is ready to run, but
- not yet admitted or dispatched for running
- Blocked: The task is ready to run, but
- not yet admitted or dispatched for running
- Running: The task is ready to run, but
- not yet admitted or dispatched for running
- Waiting for a free CPU
- Waiting for admission
- Waiting for main memory
- Swapped out of main memory
- Running, suspended
- Swapping

Communication between UNIX tasks (pipes)

For example:
```
pid = fork ();
int data_pipe [2], c, rc;

if (data_pipe [0] == -1) {
    perror ("no pipe");
    exit (1);
}
```

Languages with implicit concurrency: e.g. functional programming

- Lazy evaluation, sub-
- Concurrent program runs parallel to the user interface
- Short-circuit evaluations
- Partial functions
- Parameters are not evaluated until they are used
- lazy evaluation
- Expressions evaluated when they are needed
- nil pointer: if the evaluation of nil and the operation on it are not performed, the result is undefined
- Haskell: lazy evaluation

Concurrent programming languages

- Support for management or concurrent entities (create, terminate, ...)
- Support for communication (aoop, shared memory, ...)