Concurrency Semantics

spring 2005

What this course is about

concurrency
- "things" running in parallel, or on distributed locations
- synchronization through communication
- mobility (of code and computation, not of devices)

theory
- a calculus of static concurrent systems: CCS
- a calculus of dynamic concurrent systems: Pi
- formal syntax & operational/behavioral ***semantics***
- formal analysis and proof techniques
- equivalences & congruences

Why concurrency matters (I)

The Mars Pathfinder mission was widely proclaimed as "flawless" in the early days after its July 4th, 1997 landing on the Martian surface. Successes included its unconventional "landing" -- bouncing onto the Martian surface surrounded by airbags, deploying the Sojourner rover, and gathering and transmitting voluminous data back to Earth, including the panoramic pictures that were such a hit on the Web. But a few days into the mission, not long after Pathfinder started gathering meteorological data, the spacecraft began experiencing total system resets, each resulting in losses of data. The press reported these failures in terms such as "software glitches" and "the computer was trying to do too many things at once".

VxWorks provides preemptive priority scheduling of threads. Tasks on the Pathfinder spacecraft were executed as threads with priorities that were assigned in the usual manner reflecting the relative urgency of these tasks.

Pathfinder contained an "information bus", which you can think of as a shared memory area used for passing information between different components of the spacecraft. A bus management task ran frequently with high priority to move certain kinds of data in and out of the information bus. Access to the bus was synchronized with mutual exclusion locks (mutexes).

The meteorological data gathering task ran as an infrequent, low priority thread, and used the information bus to publish its data. When publishing its data, it would acquire a mutex, do writes to the bus, and release the mutex. If an interrupt caused the information bus thread to be scheduled while this mutex was held, and if the information bus thread then attempted to acquire this same mutex in order to retrieve published data, this would cause it to block on the mutex, waiting until the meteorological thread released the mutex before it could continue. The spacecraft also contained a communications task that ran with medium priority.

Most of the time this combination worked fine. However, very infrequently it was possible for an interrupt to occur that caused the (medium priority) communications task to be scheduled during the short interval while the (high priority) information bus thread was blocked waiting for the (low priority) meteorological data thread. In this case, the long-running communications task, having higher priority than the meteorological task, would prevent it from running, consequently preventing the blocked information bus task from running. After some time had passed, a watchdog timer would go off, notice that the data bus task had not been executed for some time, conclude that something had gone drastically wrong, and initiate a total system reset.

This scenario is a classic case of priority inversion.

HOW WAS THIS DEBUGGED?

VxWorks can be run in a mode where it records a total trace of all interesting system events, including context switches, uses of synchronization objects, and interrupts. After the failure, JPL engineers spent hours and hours running the system on the exact spacecraft replica in their lab with tracing turned on, attempting to replicate the precise conditions under which they believed that the reset occurred. Early in the morning, after all but one engineer had gone home, the engineer finally reproduced a system reset on the replica. Analysis of the trace revealed the priority inversion.

http://catless.ncl.ac.uk/Risks/19.49.html

Why concurrency matters (II)

Ever looked at the code of, e.g., the Linux Kernel ?

Why concurrency matters (III)

Between June 1985 and January 1987, a computerized radiation therapy machine called Therac-25 caused 6 known "accidents" (death of patients!) of massive radiations overdoses.

Concurrent programming errors played an important role in these accidents. "Race conditions" between different concurrent activities in the control program resulted in bad control outputs.

Because problems occurred only sporadically, they took a long time to be detected and fixed.

Why concurrency is hard ...

A very large number of possible execution histories, depending on the order in which instructions of individual processes (or: threads) are processed.

Hence, concurrent programs are hard to write and verify.

They are almost impossible to debug,
at least with standard techniques.

Rationale (I)

The ***practice*** of concurrent/communicating systems was/is:
- not a stable craft, nor a well-established science
- not equipped with a standard set of constructions (which do exist to some extent for sequential systems)

Theories usually arise to explain practice ...

The ***naive*** way: extend known sequential models
The ***foundational*** way:
- build a dedicated foundational model without "legacy"
- develop and study its theory
- analyze what arises "naturally" within this model

Rationale (II)

Two calculi

Milner 89: "Communication and Concurrency"
- closed communication protocols
- parallel computation
- *CCS*

Milner 99: "Communicating and Mobile Systems"
- open/dynamic communication protocols
- distributed and mobile computation, Internet
- *Pi Calculus*

Rationale (III)

Why communicating/mobile systems ?
- increasing number of ***existing systems***
- tend to be ***complex***
- tend to be ***error-prone***

Why calculi ?
- ***compositional***: break big things into several tiny things
- ***algebraic***: ease mechanical verification
- ***syntactic***: provide basis for programming languages

History of this course @ EPFL

2000/01 + 2001/02
Theory of Communicating & Mobile Systems
2002
Concurrency Theory
2002
Foundations of Programming (with Prof. Odersky)
2002/03 + 2003/04
Concurrency: Languages, Programming & Theory
(with Prof. Odersky)
2003 (+ 2004)
Advanced Topics on Programming Languages & Concurrency
(with Prof. Odersky)
2005 ...

Objectives

(better) understanding concurrency

analytical (mathematical) skills

presentation skills

Course Web

http://lamp.epfl.ch/
➥ Teaching

or

http://lamp.epfl.ch/~uwe/
➥ Courses