Introduction

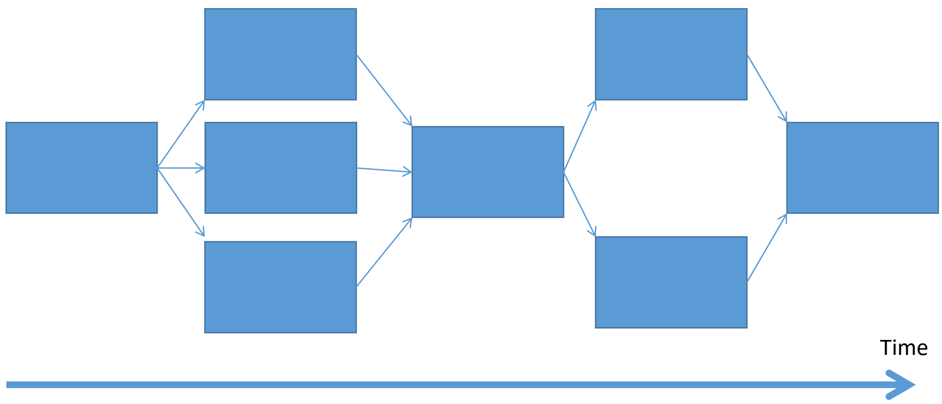
Concurrent and parallel programming
Programowanie współbieżne i równoległe
Academic year: 2018/19, Lecture 1

Paweł Lula, Cracow University of Economics, Poland
pawel.lula@uek.krakow.pl

Concurrency. Threads as a realization of the concurrency concept.

Concurrency

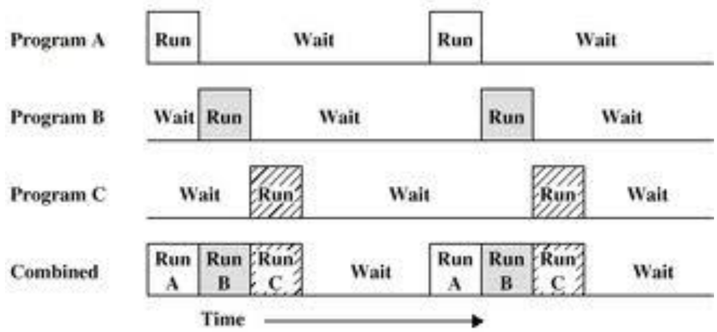
- Concurrency – a concept which assumes that several computations are executing simultaneously.



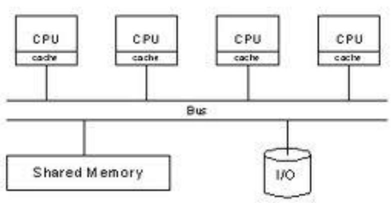
Concept of concurrency - methods of realization

- Two methods of realization:
  - time-sharing systems,
  - parallel systems.

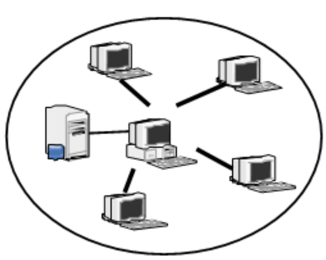
Time-sharing systems



Parallel systems



Multiprocessor system



Distributed system

Advantages of concurrency

- reduction of execution time = calculations can be performed at the same time,
- increased efficiency of computer system in time-sharing systems = better usage of computer resources (when computer is waiting for input/output operation in one program, processor can perform calculations in another program).

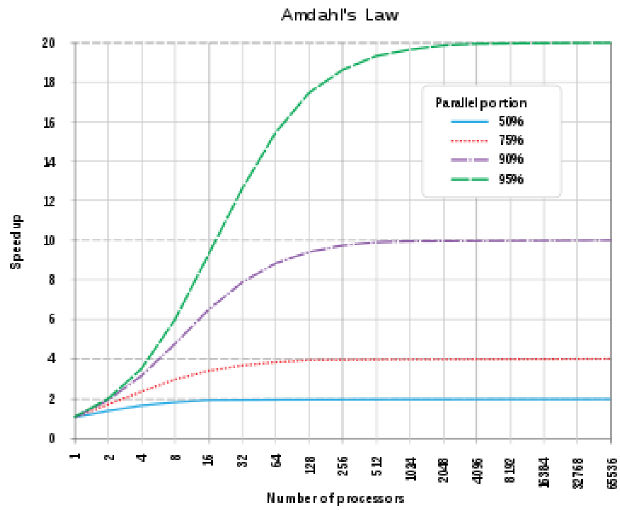
Amdahl's law (1967)

- Assumption:
  - *p* – proportion of program execution time that can be executed parallel
  - (1 – *p*) – proportion of program execution time that can not be executed parallel
  - *N* – number of processors
- Speedup:

$$S = \frac{1}{(1-p) + \frac{p}{N}}$$

- Example:
  - $p = 0{,}5$; $N = 2$ → $S = 1{,}33333$
  - $p = 0{,}5$; $N = 100$ → $S = 1{,}99$

Amdahl's law



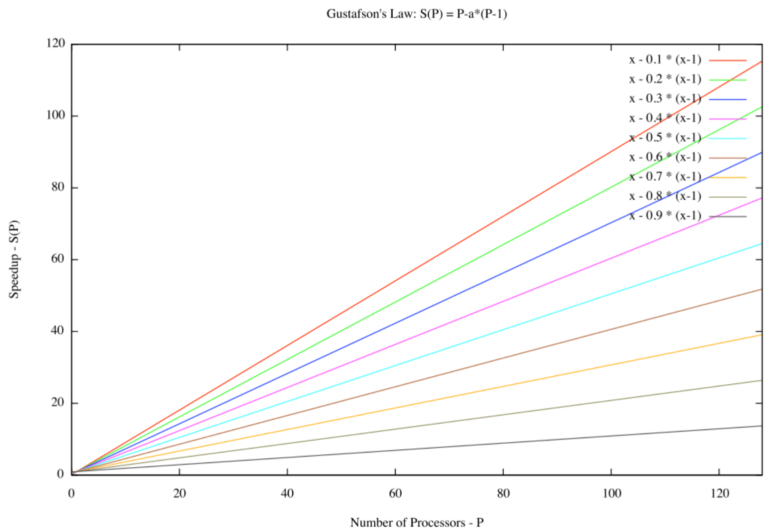
Gustafson's law (1988)

- Assumption:
  - p – proportion of program execution time that can be executed parallel
  - $(1 – p)$ – proportion of program execution time that can not be executed parallel
  - N – number of processors
- Speedup:

$$S = N - (1 - p)(N - 1)$$

- Example:
  - $p = 0,5$; $N = 2 \rightarrow S = 1,5$
  - $p = 0,5$; $N = 100 \rightarrow S = 50,5$

Gustafson's law (1988)



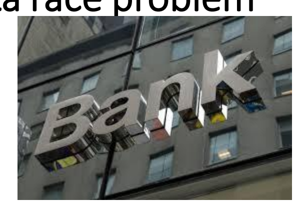
Disadvantages of concurrency

- difficulties with algorithm designing:
  - necessity of dividing the task into some subtasks which can be done simultaneously,
  - necessity of defining methods od communication between subtasks,
  - need to share common resources between subtasks (e.g.: output stream),
- increased probability of making errors during algorithm designing

Data race problem

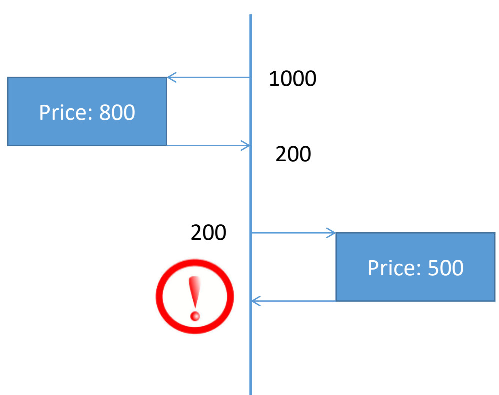
- The results of calculation depend on the way of threads execution.

Example – data race problem



Common bank account for a couple

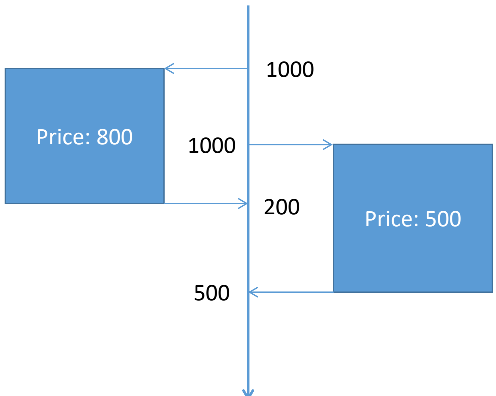
BALANCE: 1000



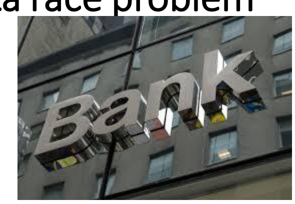
Example – data race problem

Common bank account
for a couple

BALANCE: 1000

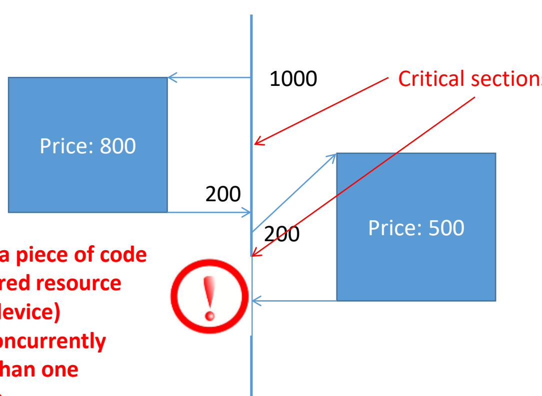


Example – data race problem



Common bank account
for a couple

BALANCE: 1000



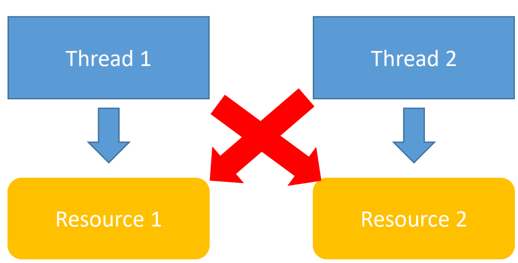
A critical section is a piece of code that accesses a shared resource (data structure or device) that must not be concurrently accessed by more than one thread of execution.

Deadlock

- Deadlock – the situation in which two threads want to get access to resources blocked by another process.

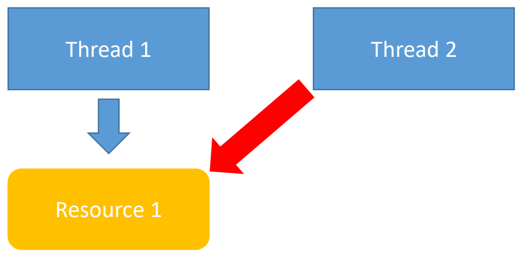
Deadlock

- Deadlock – the situation in which two threads want to get access to resources blocked by the other thread.



Starvation

- Starvation – the situation in which one thread has got the access to the common resource and does not allow the other to use it.



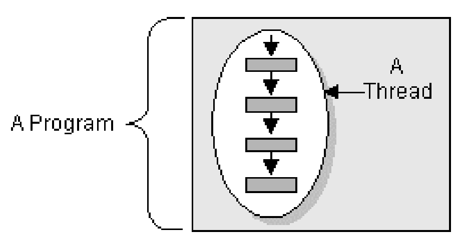
Concurrency implementation

Concurrency implementation in time-sharing systems

- Processes
  - memory allocation – computer memory is divided between processes running in the computer system,
  - memory protection – a process can only have an access to the memory which has been allocated to it
- Threads
  - common memory – computer memory is not allocated between threads; all threads use the shared block of memory (they can use common variables)
- Concurrency in Java is implemented by threads executing in time-sharing way

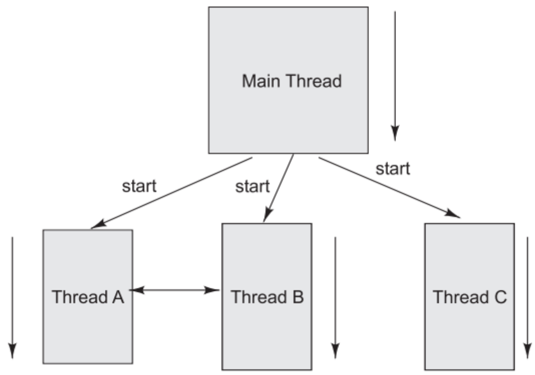
Single-thread program in Java

- The program = one thread (represented by *main* method)



program completion = completion of the main method

Multi-thread program in Java



Threads may switch or exchange data/results

program completion = completion of the main method and completion of all threads

Synchronization

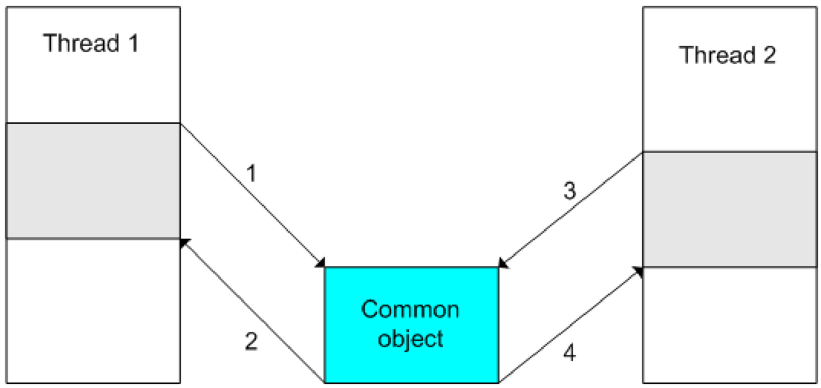
Synchronization

- Synchronization:
  - coordination of events in time;
  - harmonization of events, which allows to perform several operations properly at the same time
- Synchronization in concurrent programming = the mechanism which ensures proper results of the program regardless of the order in which actions from different tasks are executed.

Critical section

- **Critical section of a thread** – a part of the code from which common resources (variables, devices) are accessed and used.
- Only **one thread** can perform its critical section related to the same resource.

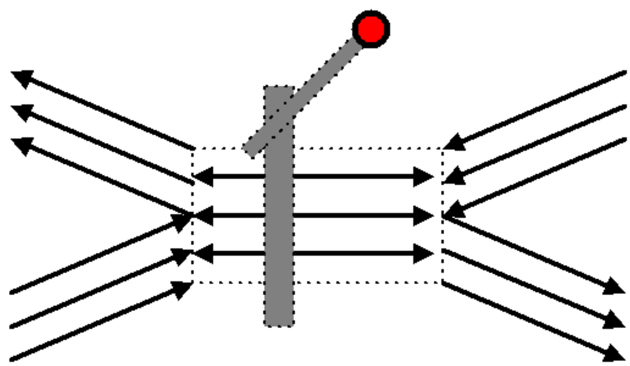
Schema of critical section



Mechanisms ensuring synchronization

- Monitors
- Semaphores

Semaphores



Semaphore – a tool used to control access to a common resource by multiple processes. Semaphore is a counter showing how many units of a common resource are available. Invented by Edsger Dijkstra in 1962.

Operations perform by semaphores

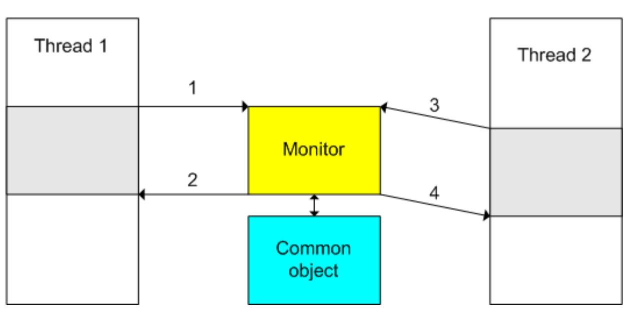
- **P** – *prolaag (Dutch) = probeer te verlagen* means ***try and decrease***. This operation is also called ***wait***.
- If a process wants to use one unit of a common resource, then it calls the P operations
  - if the counter is greater than zero ➔ the counter is decreases and the process gets the access to one unit of a common resource
  - else ➔ the processes execution is blocked until the counter value is greater than zero.

Operations perform by semaphores

- **V** – *verhoog (holenderski)* means ***increase***. This operation is also called ***signal***.
- The process release one unit of a common resource and increase the counter.

Monitor

- Monitor – a tool used for implementation of rules of access to common objects by threads.
- During the execution of critical section the thread occupies the monitor of the object related to this part of a code.
- Introduced by Per Brinch Hansen and Charles Antony Richard Hoare in 1974.



Operations perform by monitors

- ***wait*** – the thread wants to get access to the common resource:
  - if it is possible ➔ it begins to occupy the monitor,
  - else ➔ waits
- ***signal*** (***notify***) – inform all waiting threads that monitor is available

Thank you for your attention!

Paweł Lula, Cracow University of Economics, Poland
pawel.lula@uek.krakow.pl