Parallel programming and synchronization

NSWI170 Computer Systems

Jakub Yaghob, Martin Kruliš

Parallel and concurrent computing

- Parallel computing –> To je opravdu více instrukcí najednou
  - Calculations or executions of processes are carried out simultaneously
  - Bit-level, instruction-level, data, and task parallelism
  - Parallelism without concurrency – bit-level parallelism
  - Problem broken into several similar subtasks, results combined
- Concurrent computing –> To je multitasking
  - Computations are executed simultaneously
  - Concurrent without parallelism – multitasking on a single CPU
  - Processes do not work on related tasks
- Forces
  - One (shared) address space
  - Threads
  - Multiprocessing
  - Scheduling

Race condition

- Race condition
  - Multiple threads accessing (updating) the same data in shared memory space
  - Result of a computation depends on the sequence or timing of units of scheduling

```
class List {
  private:
    Node *root;

  public:
    void PushFront(Node *n) {
      n->next = root;
      root = n;
    }
};
```

Race condition

- Shared variable

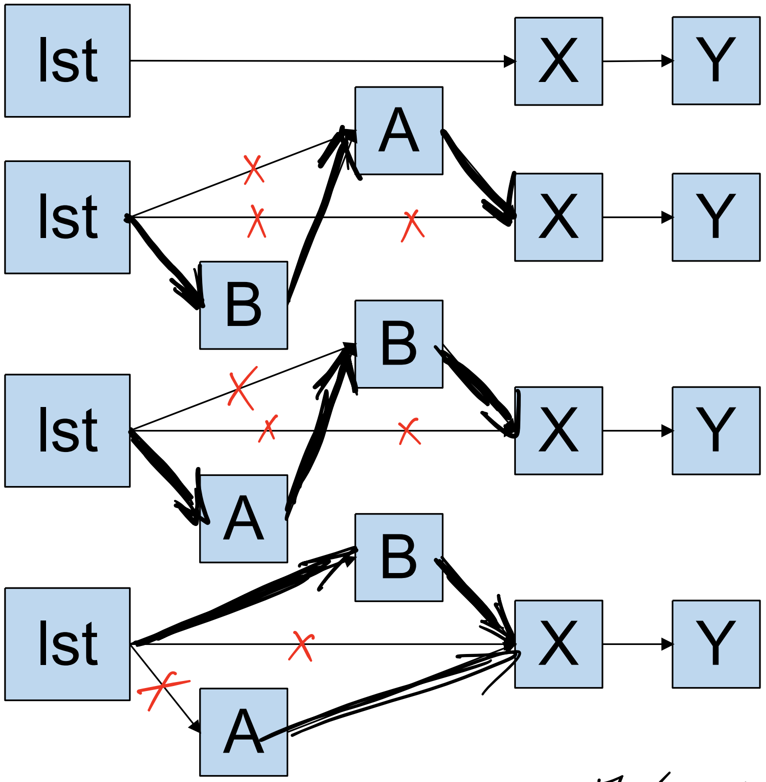
List lst;

- Thread 1

lst.PushFront(A);

- Thread 2

lst.PushFront(B);



Zápis se musí „Serializovat", tedy rozhodnu, co zapíše první a co druhý. Čtení umím paralelně.

Critical section

- Problem definition
  - Concurrent access to a shared resource can lead to the race condition or even to an undefined behavior
- Solution
  - Parts of the program, where the shared resource is accessed, need to be protected to avoid concurrent access
- Critical section
  - Protected section of the program
- Mutual exclusion
  - A critical section can be executed simultaneously by at most one unit of scheduling

Synchronization

- Process synchronization
  - Multiple units of scheduling do some form of a handshake at a certain point to make an agreement to a certain sequence of action
- Data synchronization
  - Keeping multiple copies of data in coherence with each other
  - Maintain data integrity
  - Usually implemented by process synchronization
- Problems with synchronization
  - Deadlock, starvation, …

Synchronization primitives

- Synchronization primitives
  - Implement process synchronization
  - Active
    - Instructions are executed during waiting for an access
      - Busy-waiting (testing a condition in a loop)
  - Passive/blocking
    - Unit of scheduling is blocked until the access is allowed

-> Stojí dost režie

-> Je levný, když kolize je velmi málo pravděpodobná

- Hardware support
  - Atomic instructions
    - Test-and-set (TSL), compare-and-swap (CAS)
    - Instruction semantics:

```
bool cas(T* var, T old, T newVal)
{
   if (*var != old) return false;
   *var = newVal;
   return true;
}
```

This is realized as **one instruction!**

Synchronization primitives

- Spin-lock
  - Busy-waiting using TSL/CAS
  - Short latency, right for short waiting times
- Semaphore
  - Protected counter and a queue of waiting US
  - Atomic operations UP and DOWN

```
void down() {
   if (counter>0) --counter;
   else {
        queue.push(myUS);
        myUS.block();
   }
}
```

```
void up() {
   if (counter==0 && !queue.empty()) {
      US = queue.popany();
      US.unblock().
   }
   else ++counter;
}
```

Synchronization primitives

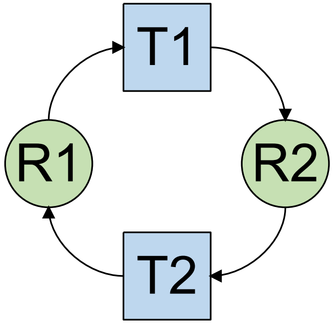
- Mutex
  - Implements mutual exclusion (semaphore with counter = 1)
  - Atomic operations LOCK and UNLOCK (corresponding to UP and DOWN)
- Barrier
  - Multiple units of scheduling meet in the same time on the same barrier
- Specific programming language constructs
  - Monitor
    - Methods in an object executed with mutual exclusion
    - Possibility to wait on a certain condition
  - Java/C#
    - Keyword **synchronized**/ **lock**

Creates a critical section

Deadlock

Deadlock

- Deadlock
  - A state of a group of units of scheduling and resources, where every member of the group waits for an action, which can be performed by other member in the group
- Necessary conditions for deadlock (Coffman)
  - Mutual exclusion
    - At least one resource in exclusive mode
  - Hold and wait
    - US holding a resource requests for another one
  - No preemption
    - Resources cannot be reclaimed without harm
  - Circular wait
    - There is a circle in a deadlock modelling graph



Deadlock – example

Shared mutexes

```
Mutex m1, m2;
```

Thread 1

```
m1.lock();
m2.lock();

m2.unlock();
m1.unlock();
```

Thread 2

```
m2.lock();
m1.lock();

m1.unlock();
m2.unlock();
```

Classic synchronization problems

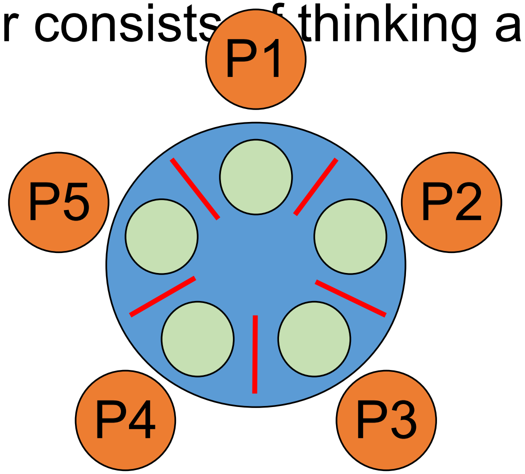
- Classic synchronization problems
  - Set of well-known synchronization problems
  - Demonstrate a problem using an allegory
  - Avoid deadlock, starvation, and other problems
- Bounded-buffer (producer-consumer)
- Dining philosophers
- Readers and writers
- Sleeping barber

Producer-consumer

- Problem
  - Producer produces a product and he places it to the warehouse with a limited capacity. If the warehouse is full, producer will stop production of products.
  - Consumer takes a product from the warehouse. If there is no item available, consumer will wait for an item.
  - If the warehouse is empty and producer produces the first product and there is a waiting consumer, producer will wake up consumer
  - If the warehouse is full and consumer takes the first product and there is stopped producer, consumer will wake up producer

Dining philosophers

- Problem
  - N philosophers sitting around a circular table
  - Each philosopher has a plate of Chinese food in front of him
  - There is one chopstick between each dish, two chopsticks are needed to eat
  - The life of a philosopher consists of thinking and eating

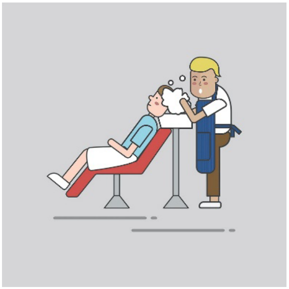


Readers and writers

- Problem
  - Common data structure
  - Readers are able only to read data
  - Writers change data or a data structure
  - Many readers read simultaneously data
  - Only one writer can change data/data structure
  - Reader must wait, if there is a working writer
  - Writer must wait, if there are working readers

Sleeping barber

- Problem
  - Barber shop with one barber, one barber chair, and N waiting chairs
  - When there is no customer, barber goes to sleep in the barber chair
  - Barber must be woken when a customer comes in
  - When barber is cutting hair, new incoming customers are waiting in chairs or leaving the shop, if there is no empty chair



Discussion