Programovanie v operačných systémoch

08 - Synchronization

Jozef Šiška

Department of Applied Informatics
Comenius University in Bratislava

2025/2026

Problems

Primitives

Memory ordering

Problems 2

Problem

A simple modification of a value in memory ,such as `ref--` is actually three steps on most architectures: fetch, modify and store back:

```
|  mov eax,ref    |                 |
|  sub eax, 1     |                 |
|  mov ref, eax   |                 |
|                 |  mov eax,ref    |
|                 |  sub eax, 1     |
|                 |  mov ref, eax   |
```

Non-trivial data structures usually have multiple members, that must be modified "consistently":

```
|  data[end] = 42  |                   |
|  end += 1        |                   |
|                  |  data[end] = 47   |
|                  |  end += 1         |
```

Problems

- Execution order, memory (data structure) consistency
- (Multiple) Read - (Multiple) Write
- Consumer - Producer
- Publish Subscribe?

- Dining Philosophers (ehm)

Primitives

- Atomic reads, writes
- TSL, CAS
- Semaphore
- Mutex (SpinLock, futext)
- Wait conditions
- Monitor
- Barriers
- IPC
- RCU / COW

Spin lock, try 1

```
void lock(int* lock) {
  while (*lock != 0) ;
  *lock = 1;
}

void unlock(int* lock) {
  *lock = 0;
}
```

Do or Do not. There is no Try.

Test and set, Compare and swap

Test and set

Write 1 and return the old value "atomically", i.e. only one caller will change it from 0 to 1.
(*tsl* instruction - test and set lock) https://en.wikipedia.org/wiki/Test-and-set

Compare and swap

Compare current and a given value, if matches, write a new value. Return the old value if successfull (compare and swap) or a boolean (compare and set).
More general than TSL, can be used to implement semaphores and lock free algorithms. https://en.wikipedia.org/wiki/Compare-and-swap

Fetch and add

Atomically incremet (and return). https://en.wikipedia.org/wiki/Fetch-and-add

Semaphore

https://en.wikipedia.org/wiki/Semaphore_(programming)

Memory ordering

- Memory ordering

```
  a = b = x = y = 0
x = 1;              y = 1;
a = y ;             b = x;
```

Memory ordering

- Memory ordering

```
  a = b = x = y = 0
x = 1;              y = 1;
a = y ;             b = x;
```

- ```
  store x, 1          store y, 1
  load  REG, y        load  REG, x
  store a, REG        store b, REG
  ```
  ```

Memory ordering

- Memory ordering

```
  a = b = x = y = 0
x = 1;              y = 1;
a = y ;             b = x;
```

```
store x, 1          store y, 1
load  REG, y        load  REG, x
store a, REG        store b, REG
```

```
load  REG, y        load  REG, x
store x, 1          store y, 1
store a, REG        store b, REG
```

Memory ordering

- Memory ordering

```
  a = b = x = y = 0
x = 1;              y = 1;
a = y ;             b = x;
```

```
store x, 1          store y, 1
load  REG, y        load  REG, x
store a, REG        store b, REG
```

```
load  REG, y        load  REG, x
store x, 1          store y, 1
store a, REG        store b, REG
```

- Sequential consistency
- Memory barrier
- Atomic instruction memory semantics
- http://preshing.com/20120515/memory-reordering-caught-in-the-act/

  http://en.cppreference.com/w/cpp/atomic

  http://en.cppreference.com/w/cpp/atomic/memory_order

  https://godbolt.org/z/afWr37j34

Problems still?

- deadlock (livelock)
- priority inversion (priority inheritance)
- efficiency
- hard to analyze

Problems still?

- deadlock (livelock)
- priority inversion (priority inheritance)
- efficiency
- hard to analyze
  - Mutual exclusion problem
    - Mutual Exclusion: Only one process/thread can be in the critical section at a time
    - Progress: No process/thread is forced to wait for an available resource
    - Bounded Waiting: No process/thread can wait forever for a resource
  - Lock free, wait free

Other solutions

- Why is it so hard?
  - It's how the hardware works...
  - Current abstractions are still "low" level
  - We use the wrong paradigms?
- Higher level apis, managers ("spooler" etc)
  - tries to hide the details for most things
  - users most probably need to understand how it works "under the hood" to use it correctly
- Concentrate on data, not code
  - Think of what needs to be done with the data / how it moves through the system, not about a sequence of steps that need to be executed
  - Qt signal / slots
  - Data flow languages
  - Immutable data (https://www.slideshare.net/Kevlin/thinking-outside-the-synchronisation-quadrant/12)

Other resources

Mutexes, ...

- memory based, thus mostly used for memory
- need more work to correctly use between processes

Other resources

- shared: printer (spooler, print server), hard drives (filesyste), sound card (mixing, pulseaudio), ...
- harder/not able to share: serial port, most character devices, access to files?
- data races: creating files and writing to them, creating temporary files
- file locking (`man flock`), advisory only (processes can still modify files)