Presentation of recent work

Analysis of synchronization algorithm

Tihon Nathan

February 6, 2023

The goal of the course

There was multiple topics to the course, although the following were the most important:

- Study the roles and functions of an operating system
- Study the concepts, problems and solutions of processes and threads
- Usage of multiprocessor machines

I like to think of this course as **a link between software and hardware**.

Goal of the project

The goal of the project was to **write and analyse** the behaviour of 3 standard synchronization algorithms:

- The Dining Philosophers problem
- The Producer-Consumer problem
- The Reader-Writer problem

with regard to **the number of threads** and **the type of synchronization primitives**.

If we break it down to its first principles, the main goal of the project was to analyse the cost of the `xchg` instruction.

Types of primitives

Primitives

The comparison of synchronization primitives was done between:

- POSIX mutexes and semaphores
- My own ASM implementation of spinlock and semaphores built on top of them

I implemented two types of spinlock:

- One based on the **test-and-set** algorithm
- The other based on the **test-and-test-and-set** algorithm

Results of the project

Observation

The importants observations made during this project is that `xchg` **is a costly operation** and we must use it cautiously.

Performance

But active primitives perform better then POSIX primitives **when the waiting time is short**, this is because POSIX primitives are "put on sleep" and needs to be waken up, which also has a cost.

Naive implementation

Simplest implementation

What most of my colleagues opted for is to duplicate the code and write each of the algorithms separately.

While fast/easy to write, this method comes with a strong downside:

Readability/extensibility

This method makes it very hard to navigate, read and extend the code.

Extensible implementation

The extensible implementation I opted for relies on the use of **unions nested in structs**

```
1 typedef struct {
2   algo_t type; // Type of mutex: 0=>POSIX, 1=>TAS, 2=>TATAS
3   union {
4     pthread_mutex_t* posix;
5     spinlock_t* spinlock;
6   }; // Use the type of mutex corresponding to type
7 } mutex_t;
8
```