**Exercise 2: Bottlenecks**

**Part 1: Mutex and Channel basics**

### **What is an atomic operation?**

Atomic operations cannot be divided, either they happen in whole or they do not happen at all. In programming, they have the added feature of locking the other threads, so only one thread can execute an atomic operation at a time.

These atomic operations can be used to create bigger atomic operations.

### **What is a semaphore?**

A semaphore is simply a variable used to control access to a common resource by multiple threads in a concurrent system, such as a multitasking OS.

### **What is a mutex?**

A **mut***ual***ex***clusion (mutex)*is a object that allows multiple threads to share the same resource (peripherals, file access…), but not simultaneously. Then, any thread that needs the resource must lock the mutex from other threads while it is using the resource. The mutex is unlocked when the data is no longer needed or the function is finished.

### **What is the difference between a mutex and a binary semaphore?**

Mutex can be released only by the thread that had acquired it, while binary semaphore can be released from any other thread. Thus semaphores are more suitable for some synchronization problems.

### **What is a critical section?**

A Critical Section is the part of a program that accesses shared resources. Many threads inside such a section could create race conditions.

### **What is the difference between race conditions and data races?**

A data race occurs when 2 instructions from different threads access the same memory location, at least one of these accesses is a write and there is no synchronization between them.

A race condition is a flaw that occurs when the timing or ordering of events affects a program’s correctness. Many race conditions can be caused by data races, but this is not necessary.

### **List some advantages of using message passing over lock-based synchronization primitives.**

With message passing it is easier to build parallel hardware. This is because this model is quite tolerant of higher communication latencies. It is also much easier to implement than the shared memory model.

As inconvenient, lock-based synchronization primitives (shared memory model) may create problems such as synchronization and memory protection that need to be addressed. Lock-based could also create race conditions and deadlocks.

### **List some advantages of using lock-based synchronization primitives over message passing.**

With lock-based synchronization primitives (shared memory model) is memory communication faster as compared to the message passing model on the same machine.

As inconvenient, message passing model has slower communication than the shared memory model because the connection setup takes time.