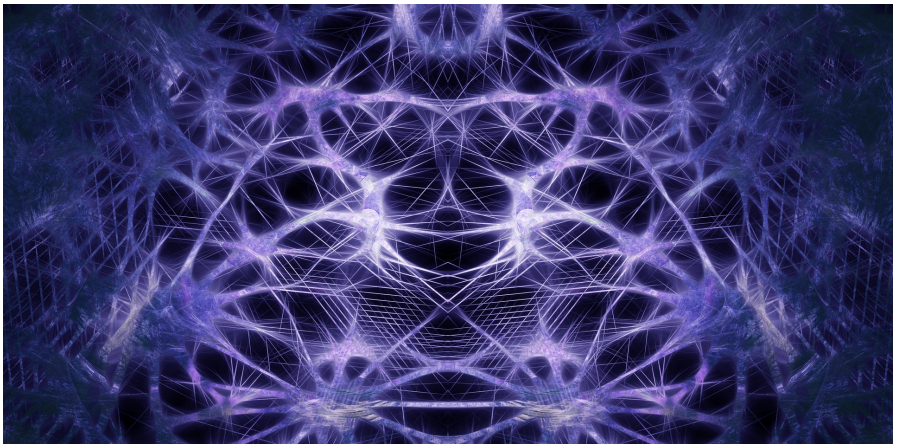

# Übung 03

Nichtsequentielle und Verteilte Programmierung

Sebastian Faase, Lutz Schäfer

Tutorium Leon Dirmeier

Freie Universität Berlin

6. Mai 2019

# I Sicherung des kritischen Abschnitts in C (10 Punkte)

Auf Grund der von uns gewählten Architektur des Programmes aus dem vorangegangenen Übungsbogen, beschränkt sich die Implementierung dieser Aufgabe auf das Hinzufügen eines weiteren *switch case m*.

Listing I.1: world.c⟶mutex

```
case 'm':
  pthread_mutex_lock(&m_lock);
  for (size_t i = 0; i < BRIDGE_LEN; i++)
  {
    drive(i,&sleep_timer,&sleep_remainder);
  }
  pthread_mutex_unlock(&m_lock);
  break;
```

Listing I.2: output1

```
./sim -l m -n 150 -r 100000
Car 0 pthread id 0x0007d9a8077f0000 rolling.
Car 3 pthread id 0x00d758a7077f0000 rolling.
Car 2 pthread id 0x00e7d8a7077f0000 rolling.
Car 4 pthread id 0x00c7d8a6077f0000 rolling.
[...]
Car 8F pthread id 0x0097f75c077f0000 rolling.
Car 90 pthread id 0x0087775c077f0000 rolling.
Car 91 pthread id 0x0077f75b077f0000 rolling.
Car 92 pthread id 0x0067775b077f0000 rolling.
Car 93 pthread id 0x0057f75a077f0000 rolling.
Car 94 pthread id 0x0047775a077f0000 rolling.
Car 95 pthread id 0x0037f759077f0000 rolling.
there were 0 collisions
```

# II Sicherung des kritischen Abschnitts in (8 Punkte)

Wie haben Performance als zweites Vergleichskriterium gewählt.

## (1) Korrektheit

Da wir für mehr als zwei Autos nur den Algorithmus von Lamport nutzen, sind beide Varianten gleich korrekt, da es sich bei Beiden um Prioritätswarteschlangen implementierende Schlossalgorithmen handelt.

## (2) Performance

Um die Performance der beiden Algorithmen besser vergleichen zu können, betrachten wir die zwei möglichen Szenarien der beiden Algorithmen.

### (2.1) Schloss ist offen, Warteschlange leer

In diesem Fall ist kein Unterschied festellbar (Mutex hat minimalen overhead und Lamport evtl Pipeline flush).

### (2.2) Schloss ist offen, Warteschlange voll

Im Falle des Mutex' findet ein Kontextwechsel statt und der das Mutex haltende Prozess wird direkt (in Abhängigkeit des Betriebssystems) aufgeweckt. Dies spart gegenüber dem busywaiting (Lamport) CPU-Zyklen, da der nächste gewählte Prozess jener ist, welcher das Schloss erwirbt.

### (2.3) Schloss ist geschlossen

Im Falle des Mutex' findet ein Kontextwechsel statt und der das Mutex haltende Prozess wird direkt (in Abhängigkeit des Betriebssystems) aufgeweckt. Dies spart gegenüber dem busywaiting (Lamport) CPU-Zyklen, da der nächste gewählte Prozess jener ist, welcher das Schloss freigibt.

# III Modellierung mit Petri-Netzen (8 Punkte)