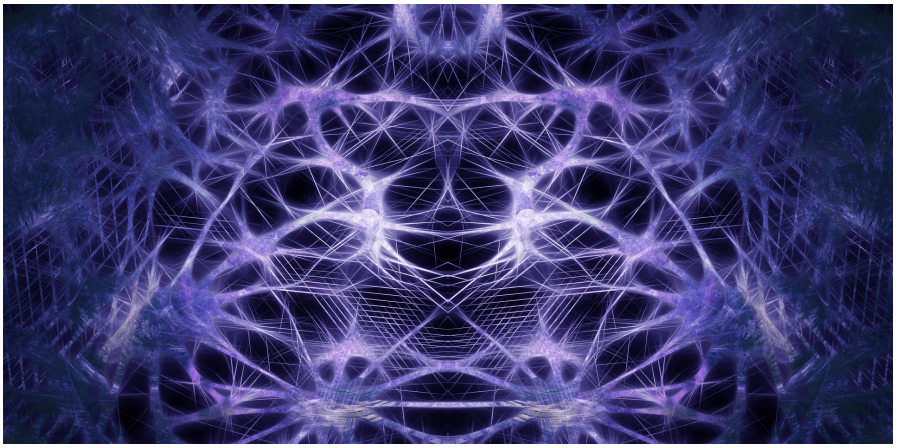

Übung 02

Nichtsequentielle und Verteilte Programmierung

Sebastian Faase, Lutz Schäfer

Tutorium Christopher Filsinger (Mo)

Freie Universität Berlin

2. Mai 2019

I Kritischer Abschnitt in C (10 Punkte)

Um unsere Implementierung mit den Vorgaben dieser Aufgabe zu starten, nutzten wir

```
./sim -l n -n 2 -r 100000
```

alternativ auch

```
./sim
```

da die Parameter als *default* gewählt wurden.

Die Simulation hat einige Kommandozeilenparameter, die wir in der Simulation der Welt behandeln [I.1]. Die Welt, in der sich die Brücke mit den Autos befindet, nutzt globale Variablen [I.2], definiert zwei Strukturen [I.3] und implementiert neben der Welt selbst [I.4] noch die Funktionen für das Verhalten der Autos [I.5], ihr Befahren der Brücke [I.6], ob es auf der Brücke Kollisionen gab [I.7], so wie einige Hilfsfunktionen zur Visualisierung der Brücke [I.8][I.9][I.10].

Da unsere Ausgabe hundert Zeilen pro Sekunde erzeugt, sei hier nur ein Auszug der Ausgabe Wiedergegeben [I.11].

Listing I.1: simulation.c

```
int main(int argc, char *argv[])
{
  int64_t help_is_set = 0;
  worldParameter parameter = {.algorithm = 'n', .num_of_obj = 2,
                              .num_of_runs = 2};

  /* parsing commandline arguments */
  int64_t opt;
  while ( -1 != (opt = getopt(argc, argv, "hl:n:r:")) )
  {
    switch (opt) {
      case 'h':
        printhelp(stdout, argv[0]);
        help_is_set = 1;
        break;
      case 'l':
        /* use the first letter of an algorithm as its unique */
        if ('l' == optarg[0] || 'p' == optarg[0] ||
            'd' == optarg[0] || 'n' == optarg[0])
        {
          parameter.algorithm = optarg[0];
          break;
        }
        else
        {
          printhelp(stderr, argv[0]);
          exit(EXIT_FAILURE);
        }
      case 'n':
        parameter.num_of_obj = atoi(optarg); // TODO use strtol
        break;
```

```
      case 'r':
        parameter.num_of_runs = atoi(optarg);
        break;
      default: /* .?. */
        printhelp(stderr, argv[0]);
        exit(EXIT_FAILURE);
    }
  }

  /* check for inconsistend parameter */
  if ( 2 != parameter.num_of_obj
      && ( 'd' == parameter.algorithm || 'p' == parameter.algorithm ) )
    parameter.num_of_obj = 2; // bold move to know it better

  /* if no help shall be displayed, start the world */
  if (!help_is_set)
  {
    /* setup memory for the locks */
    locks_init(parameter.num_of_obj);
    run_world(&parameter);
    /* free memory */
    locks_free();
  }

  exit(EXIT_SUCCESS);
}
```

Listing I.2: world.c⟶globals

```
uint64_t FSALP42019_WORLD_BRIDGE[BRIDGE_LEN]; // > 0 implies crash
size_t COLLISION;                             // count collisions
size_t RUN_MAX;
```

Listing I.3: world.c⟶structures

```
typedef struct {
  uint64_t * plate;
  side side;
  uint64_t pos;
  char algorithm;
} carArgs;

typedef struct {
  pthread_t car;
  int retv;
  carArgs args;
} pCar;
```

Listing I.4: world.c⟶run_world

```
void run_world(worldParameter * param)
```

```
{
  pthread_t          gfx;
  pCar               cars[param->num_of_obj];
  uint64_t           numbers[param->num_of_obj];
  for (size_t i = 0; i < param->num_of_obj; i++)                    // give the cars asc
  {
    numbers[i]=i;
  }
  RUN_MAX = param->num_of_runs;
  srand(time(NULL));                                                // init random seed

  COLLISION = 0;                                                    // init collision co
  for (size_t i = 0; i < BRIDGE_LEN; i++)                           // init empty bridge
    FSALP42019_WORLD_BRIDGE[i] = 0;

  int retshow = 0;
  if ( 0 != (retshow = pthread_create(&gfx, NULL, show_world, NULL) ))
    exit(EXIT_FAILURE);

  for (size_t i = 0; i < param->num_of_obj; i++)                    // spawn the cars
  {
    cars[i] = (pCar) {
      .args.plate = &numbers[i],
      .args.side = (side) (rand() % 2),
      .args.algorithm = param->algorithm
    };
    cars[i].retv = pthread_create(&cars[i].car, NULL, drive_car,
                                  (void*) &cars[i].args);
    if ( cars[i].retv )                                             // use retv for erro
      printf("Car Number %ld is not functional (ErrorCode: %d)\n",
             i, cars[i].retv);
  }

  for (size_t i = 0; i < param->num_of_obj; i++)                    // ensure cars are d
    pthread_join(cars[i].car, NULL);

  // pthread_join(retshow, NULL);
  bridge_status();
  printf("there were %ld collisions\n", COLLISION);
  exit(0);
}
```

Listing I.5: world.c⟶drive_car

```
void * drive_car(void * args)
{
  struct timespec sleep_timer, sleep_remainder;                     // to stay on bridge
  sleep_timer.tv_nsec = (rand() % 10) * 100000000L;                 // 10th - 100th seco
  sleep_timer.tv_sec  = 0;
  carArgs * cargs = (carArgs *) args;                               // retriving argume
```

```
  uint64_t * number = cargs->plate;
  char  algorithm = cargs->algorithm;
  pthread_t thread_id = pthread_self();
  print_thread_id(thread_id, *number);


  for (size_t r = 0 ; r < RUN_MAX; r++)
  {
        switch (algorithm)
        {
          case 'n':
            for (size_t i = 0; i < BRIDGE_LEN; i++)
            {
              drive(i,&sleep_timer,&sleep_remainder);
            }
            break;
          case 'l':
            lock_lamport(*number);
            for (size_t i = 0; i < BRIDGE_LEN; i++)
            {
              drive(i,&sleep_timer,&sleep_remainder);
            }
            unlock_lamport(*number);
            break;
          case 'd':
            lock_dekker(* number);
            for (size_t i = 0; i < BRIDGE_LEN; i++)
            {
              drive(i,&sleep_timer,&sleep_remainder);
            }
            unlock_dekker(* number);
            break;
          case 'p':
            lock_peterson(* number);
            for (size_t i = 0; i < BRIDGE_LEN; i++)
            {
              drive(i,&sleep_timer,&sleep_remainder);
            }
            unlock_peterson(* number);
            break;
        }
        /*********/
  }
  return NULL;
}
```

Listing I.6: world.c⟶drive

```
void drive(size_t i,
           struct timespec * sleep_timer,
```

```
            struct timespec * sleep_remainder)
{
  ++FSALP42019_WORLD_BRIDGE[i];
  nanosleep(sleep_timer, sleep_remainder);                    // as above, to make
  bridge_status();
  --FSALP42019_WORLD_BRIDGE[i];
}
```

Listing I.7: world.c⟶bridge_status

```
int64_t bridge_status(void)
{
  for (size_t i = 0; i < BRIDGE_LEN; i++)
  {
    uint64_t tmp = FSALP42019_WORLD_BRIDGE[i];
    // check for collision
    if (tmp > 1)
      ++COLLISION;
  }
  return 0;
}
```

Listing I.8: world.c⟶show_world

```
void * show_world(void * arg)
{
  (void) arg; // UNUSED
  struct timespec sleep_timer, sleep_remainder;
  sleep_timer.tv_nsec = 1000000L;                             // 10th - 100th seco
  sleep_timer.tv_sec  = 0;

  while (1)
  {
    print_bridge();
    nanosleep(&sleep_timer, &sleep_remainder);
  }
  return NULL;
}
```

Listing I.9: world.c⟶print_thread_id

```
void print_thread_id(pthread_t thread_id, uint64_t number)
{
  unsigned char * pthread_id = (unsigned char*)(void*)(&thread_id);  // get pointer t
  uint64_t buffer_position        = 0;
  char buffer[sizeof(thread_id)+40];                          // 1 byte = 2 hex ch

  buffer_position  = sprintf(buffer+buffer_position,
                             "Car␣%"PRIX64"␣pthread␣id␣0x", number);
  for (size_t i = 0; i < sizeof(thread_id); i++)
    buffer_position += sprintf(buffer+buffer_position, "%02x",
```

```
                                        (unsigned)(pthread_id[i]));
  buffer_position += sprintf(buffer+buffer_position, " rolling.\n");
  buffer[buffer_position] = 0;
  printf("%s", buffer);
}
```

Listing I.10: world.c⟶print_bridge

```
void print_bridge(void)
{
  uint64_t buffer_position = 0;
  char buffer[BRIDGE_LEN*30];

  buffer_position  = sprintf(buffer+buffer_position, "=");
  for (size_t i = 0; i < BRIDGE_LEN; i++)
  {
    uint64_t tmp = FSALP42019_WORLD_BRIDGE[i];
    // check for collision
    if (tmp > 1)
    {
      buffer_position += sprintf(buffer+buffer_position, "x=");
    }
    else
    {
      buffer_position += sprintf(buffer+buffer_position,
                                 "%"PRIx64"=", tmp);
    }
  }

  buffer_position += sprintf(buffer+buffer_position, "\n");
  buffer[buffer_position] = 0;
  printf("%s", buffer);
}
```

Listing I.11: output1

```
./sim -l n -n 2 -r 100000
Car 0 pthread id 0x0007c0f05a7f0000 rolling.
Car 1 pthread id 0x00f73ff05a7f0000 rolling.
there were 79351 collisions
```

II Sicherung des kritischen Abschnitts in C (8 Punkte)

Wir haben auf Grund der ungenauen Aufgabenstellung alle in Frage kommenden Lock-Algorithmen implementiert. Diese sind Dekker, Petterson und Lamport. Es ist zu beachten, dass Dekker nur mit maximal zwei Fäden korrekt arbeitet.

Um das Programm mit den jeweiligen Algorithmen zu starten, muss der Parameter -l entsprechend gewählt werden. So startet folgender Aufruf z.B. mit dem Algorithmus von Lamport.

```
./sim -l l -n 2 -r 100000
```

Da unsere Ausgabe hundert Zeilen pro Sekunde erzeugt, sei hier nur ein Auszug der Ausgabe Wiedergegeben [II.1].

Listing II.1: output1

```
./sim -l l -n 2 -r 100000
Car 0 pthread id 0x0007c0f05a7f0000 rolling.
Car 1 pthread id 0x00f73ff05a7f0000 rolling.
there were 0 collisions

./sim -l d -n 2 -r 100000
Car 0 pthread id 0x0007c0f05a7f0000 rolling.
Car 1 pthread id 0x00f73ff05a7f0000 rolling.
there were 0 collisions

./sim -l p -n 2 -r 100000
Car 0 pthread id 0x0007c0f05a7f0000 rolling.
Car 1 pthread id 0x00f73ff05a7f0000 rolling.
there were 0 collisions
```

Hier noch die Algorithmen von Dekker [II.2], Lamport [II.3] und Peterson. [II.4]

Listing II.2: locks.c⟶dekker

```
/*************************************************************/
/* lock critical section with dekker algorithm               */
/* this works only for 2 Processes                           */
void lock_dekker(uint64_t p_id)
{
  ENTERING[p_id] = 1;
  while ( ENTERING[NUMBER_OF_OBJECTS - 1 - p_id] ) {          //
    if (FAVOURED != p_id) {
      ENTERING[p_id] = 0;
      while( FAVOURED != p_id )
        ; // busy waiting
      ENTERING[p_id] = 1;
    }
  }
}

/*************************************************************/
/* unlock critical section with dekker algorithm             */
void unlock_dekker(uint64_t p_id)
```

```
{
  FAVOURED = NUMBER_OF_OBJECTS - 1 - p_id;
  ENTERING[p_id] = 0;
}
```

Listing II.3: locks.c⟶lamport

```
void lock_lamport(uint64_t p_id)
{
  /* set the priority of our process  */
  ENTERING[p_id] = 1;
  PRIORITAET[p_id] = 1 + max_of_array(PRIORITAET, NUMBER_OF_OBJECTS);
  ENTERING[p_id] = 0;

  /* look if another proccess has to be processed first        */
  for (size_t i = 0 ; i < NUMBER_OF_OBJECTS; i++)
  {
    while(ENTERING[i])
      ;     // busy waiting until another process retrives priority
    while(PRIORITAET[i] != 0 && PRIORITAET[i] < PRIORITAET[p_id] )
      ;     // busy waiting until another process with lower priority finished
  }
}

/*************************************************************/
/* unlock critical section with lamport algorithm            */
void unlock_lamport(uint64_t p_id)
{
  PRIORITAET[p_id] = 0;

}
```

Listing II.4: locks.c⟶peterson

```
void lock_peterson(uint64_t p_id)
{
  ENTERING[p_id] = 1;
  FAVOURED = NUMBER_OF_OBJECTS - 1 - p_id;
  while ( ENTERING[NUMBER_OF_OBJECTS - 1 - p_id]
       && FAVOURED == (NUMBER_OF_OBJECTS - 1 - p_id) )
    ;             // busy waiting for another thread to finish

}

/*************************************************************/
/* unlock critical section with peterson algorithm           */
void unlock_peterson(uint64_t p_id)
{
  ENTERING[p_id] = 0;
}
```

III Erweiterung der Anzahl der Threads in C (8 Punkte)

Diese Aufgabe haben wir berits für die Algorithmen von Lamport und Peterson geschenkt bekommen, mit Dekker ist sie nicht möglich. Wir ändern einfach den wert des Parameters -n auf eine Zahl > 2.

```
./sim -l l -n 100 -r 100000
```

Da unsere Ausgabe hundert Zeilen pro Sekunde erzeugt, sei hier nur ein Auszug der Ausgabe Wiedergegeben [III.1].

Listing III.1: output1

```
./sim -l l -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
```

IV Bewertung der Ansätze (4 Punkte)

Das OS kann die Prozesse in Abhängingkeit des Schedulars in beliebiger Reihenfolge abarbeiten. Insbesondere kann es busywaiting erkennen und den Prozess schlafen legen. Dadurch kann ein anderer Prozess vorgezogen werden. Da wir die Reihenfolge des interleavings nicht betrachten ist das nicht relevant. Um diese Effekte zu vermeiden, kann sowohl die Priorität als auch der Scheduling Algorithmus der Prozesse verändert werden. (siehe man 3 pthread_attr_init)