Barry Linnert

Nichtsequentielle und verteilte Programmierung, SS2021

Übung 3

Tutor: Florian Alex
Tutorium 3

Rui Zhao, William Djalal, Simeon Vasilev

8. Mai 2021

1 Sicherung des kritischen Abschnitts in C (10 Punkte)

Implementieren Sie die Aufgabenstellung des Übungsblatts 2, Aufgabe 3 nun mit Hilfe von POSIX-Mutex. Dokumentieren Sie Ihr Programm und stellen Sie immer die Ausgaben des jeweiligen Programms zur Verfügung.

```c
#include <pthread.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>


#define TRUE 1
#define FALSE 0

#define NUM_THREADS 5

int crash = 0;

int car[NUM_THREADS] = {0};     // 1: on bridge , 0: not on bridge

pthread_mutex_t lock;


void *cross_bridge (void *threadid)
{
 long tid;
 tid = (long) threadid;
 int _error = 0;

 for (int i = 0; i < 100000; i++)
 {
  _error = pthread_mutex_lock(&lock);  //enter the critical section
  if (_error) {
    printf ("ERROR; return code from pthread_mutex_lock() is %d\n", _error);
    exit (-1);
   }

  car[tid] = 1;       //critical section start

  int sum = 0;       //how many cars on the bridge
  for (int i = 0; i < NUM_THREADS; i++)
  {
```

```
38      sum = sum + car[i];
39    }

41    if (sum > 1)        //if more than one car, crash
42    {
43     crash++;
44    }
45
46    nanosleep((const struct timespec[]){{0, 100L}}, NULL); //time to across the bridge

48    car[tid] = 0;

50    _error = pthread_mutex_unlock(&lock); //leave the critical section
51    if (_error) {
52      printf ("ERROR; return code from pthread_mutex_unlock() is %d\n", _error);
53      exit (-1);
54     }
55  }
56
57    // return crash;
58    pthread_exit (NULL);
59  }

61  int main (/*int argc, char *argv[]*/)
62  {
63    pthread_t threads[NUM_THREADS];
64    int rc;
65    long t;

67    // init lock
68    pthread_mutex_init(&lock, NULL);

70    // creating threads
71    for (t=0; t < NUM_THREADS; t++) {
72     printf ("In main: creating thread %ld\n", t);
73     rc = pthread_create (&threads[t], NULL, cross_bridge, (void *)t);
74     if (rc) {
75      printf ("ERROR; return code from pthread_create () is %d\n", rc);
76      exit (-1);
77     }
78    }
79
80    // joining threads
81     for (long t = 0; t < 2; t++) {
82      pthread_join (threads[t], NULL);
83     }

85     // output results
86     printf("numbers of crashes :%d\n", crash);

88    /* Last thing that main() should do */
89    pthread_exit(NULL);
90  }
```

```
zhaor@6P0ONT1-88WD3YW:~/alp4/u3$ gcc -std=c11 -Wall -Wextra -pedantic -pthread a1.c -o a1
zhaor@6P0ONT1-88WD3YW:~/alp4/u3$ ./a1
In main: creating thread 0
In main: creating thread 1
In main: creating thread 2
In main: creating thread 3
In main: creating thread 4
numbers of crashes :0
zhaor@6P0ONT1-88WD3YW:~/alp4/u3$
```

10/10

Abbildung 1: Test Screen A1

2 Sicherung des kritischen Abschnitts in C (8 Punkte)

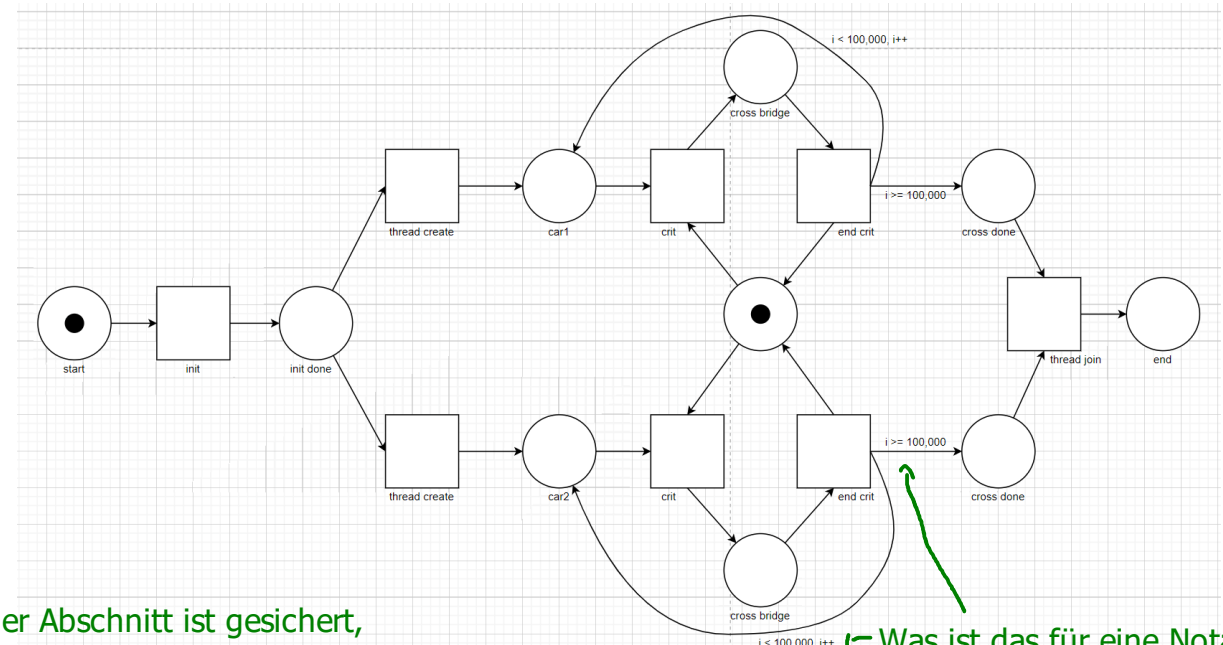
Vergleichen Sie beide Lösungen (Übungsblatt 2, Aufgabe 3 und Übungsblatt 3, Aufgabe 1) in Hinblick auf Korrektheit und (mindestens) einem weiteren, selbstgewähltem Kriterium.

- Vergleich hinsichtlich Korrektheit: Sowohl die Lösung vom letzten Übungszettel mit Multiple Locks with Mutual Access – Peterson Algorithmus, auch die jetzige mit POSIX Mutex, funktioniert nach Definition aus der Vorlesung korrekt. Ja? Siehe ab 5-60
  Solange der Posix-Standard korrekt ist, funktioniert die aktuelle Übungsblattlösung ordnungsgemäß. Das Programm läuft im kritischen Bereich sequentiell ab. Dadurch kann die partielle Korrektheit mit Hoare bewiesen werden. Weil es keine Eingabedaten gibt, folgt daraus auch die totale Korrektheit. Was sind unsere Vorbedingungen? ?
- Vergleich hinsichtlich Fairness: Der Mutex kann Fairness garantieren, daher kann die Lösung mithilfe von POSIX Mutex als fair angesehen werden. Multiple Locks with Mutual Access – Peterson Algorithmus kann auch Fairness garantieren, da im Algorithmus Warteschlangen vorhanden sind, die nach Priorität angeordnet sind. Wenn ein Auto die Brücke überquert, tritt es erneut in die Prioritätswarteschlange ein und wartet in der Warteschlange. Jedes Auto hat die gleiche Chance, die Brücke zu überqueren. Wieso sind pthread Mutexe fair?

6/8

3 Modellierung mit Petri-Netzen (12 Punkte)

Modellieren Sie Ihre Lösung der Aufgabe 1 als Petri-Netz.



kritischer Abschnitt ist gesichert,
aber die 100.000 Durchläufe sind nicht richtig modelliert

Was ist das für eine Notation?

Abbildung 2: Petri-Netz [1]

8/12

Literatur

[1] Erstellt mit hilfe von draw.io

24/30