Barry Linnert

Nichtsequentielle und verteilte Programmierung, SS2021

Übung 4

Tutor: Florian Alex
Tutorium 3

Rui Zhao, William Djalal, Simeon Vasilev

15. Mai 2021

1 Speisende Philosophen in C (12 Punkte)

Implementieren Sie in C eine Simulation des Problems der speisenden Philosophen. Die Anzahl der beteiligten Philosophen soll beliebig, aber fest gewählt werden können. Der Zustand der Philosophen (essend, denkend) soll bei jeder Änderung ausgegeben werden. Dokumentieren Sie Ihr jeweiliges 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 MAX 1000
// we assume that there are no more than 1000 philosophers on the same table.

int NUM_THREADS;
// how many philosophers

pthread_mutex_t chops[MAX];
// number of philosopher = number of chopstick,
// the rest of them are not available
// because of (right hand chopstick) = (i + 1) % Number of philosophers.


void *philosopher (void *threadid)
{
 long tid;
 tid = (long) threadid;
 int left = tid;
 int right = (tid + 1) % NUM_THREADS;

 while (1) {
  printf("philosopher %ld is thinking.\n", tid);

  printf("philosopher %ld is hungry and going to eat.\n", tid);

  pthread_mutex_lock(&chops[left]);
  printf("philosopher %ld get chopsticks %d.\n", tid, left);
  sleep(1);
  pthread_mutex_lock(&chops[right]);
  printf("philosopher %ld get chopsticks %d.\n", tid, right);

  printf("philosopher %ld is eating.\n", tid);
```

```
37
38    pthread_mutex_unlock(&chops[left]);
39    sleep(1);
40    pthread_mutex_unlock(&chops[right]);
41   }
42
43    pthread_exit (NULL);
44  }


47  int main (/*int argc, char *argv[]*/)
48  {
49   printf( "How many philosophers? :");
50      scanf("%d", &NUM_THREADS);

52    pthread_t threads[NUM_THREADS];
53    int rc;
54    long t;

56    // init lock
57    pthread_mutex_init(&chops[NUM_THREADS], NULL);

59    // creating threads
60    for (t=0; t < NUM_THREADS; t++) {
61     rc = pthread_create (&threads[t], NULL, philosopher, (void *)t);
62     if (rc) {
63      printf ("ERROR; return code from pthread_create () is %d\n", rc);
64      exit (-1);
65     }
66    }
67
68    // joining threads
69     for (long t = 0; t < NUM_THREADS; t++) {
70      pthread_join (threads[t], NULL);
71     }

73    /* Last thing that main() should do */
74    pthread_exit(NULL);
75  }
```

Array kann auch erst nach der Eingabe erstellt werden

Die Funktion philosopher() repräsentiert die Aktivität eines Philosophen, der als n verschiedene Threads angelegt werden kann, die n verschiedene Philosophen repräsentieren. Jeder Philosoph denkt zuerst, und wenn ein bestimmter Philosoph hungrig ist, nimmt er sein linkes Stäbchen auf, dann sein rechtes Stäbchen, dann isst er, dann legt er sein linkes und rechtes Stäbchen ab und denkt wieder.
Da Stäbchen eine kritische Ressource sind, wird, wenn ein Philosoph seine linken und rechten Stäbchen in die Hand nimmt, eine Lock auf beide Stäbchen gelegt, damit kein anderer Philosoph sie benutzen kann.
Erst wenn dieser Philosoph mit dem Essen fertig ist und seine Stäbchen abgelegt hat, wird die Ressource freigeschaltet und steht damit anderen Philosophen zur Verfügung.

✓

12/12

2 Verklemmungen (8 Punkte)

Kann es bei Ihrer Implementierung der Aufgabe 1 zu einer Verklemmung (deadlock) kommen? Wenn ja, legen Sie dar, welche Abfolge von Operationen zu einer Verklemmung führen kann. Wenn nein, begründen Sie an Hand Ihres Ansatzes, warum es zu keiner Verklemmung kommen kann und bewerten Sie Ihren Ansatz mindestens in Bezug auf Fairness gegenüber allen Philosophen.

```
zhaor@6P0ONT1-88WD3YW:~/alp4/u4$ gcc -std=c11 -Wall -Wextra -pedantic -pthread a1.c -o a1
zhaor@6P0ONT1-88WD3YW:~/alp4/u4$ ./a1
How many philosophers? :5
philosopher 0 is thinking.
philosopher 0 is hungry and going to eat.
philosopher 0 get chopsticks 0.
philosopher 3 is thinking.
philosopher 3 is hungry and going to eat.
philosopher 3 get chopsticks 3.
philosopher 2 is thinking.
philosopher 2 is hungry and going to eat.
philosopher 2 get chopsticks 2.
philosopher 4 is thinking.
philosopher 4 is hungry and going to eat.
philosopher 4 get chopsticks 4.
philosopher 1 is thinking.
philosopher 1 is hungry and going to eat.
philosopher 1 get chopsticks 1.
```

Abbildung 1: Verklemmung (deadlock)

Bei Implementierung der Aufgabe 1 kann es zu einer Verklemmung (deadlock) kommen.
Zuerst nimmt jeder Philosoph sein linkes Stäbchen, dann versucht jeder Philosoph, sein rechtes Stäbchen zu nehmen, und da jedes Stäbchen bereits besetzt ist, kann jeder Philosoph sein rechtes Stäbchen nicht nehmen und muss warten, bis es von einem anderen Philosophen freigegeben wird. Somit warten alle n Threads darauf, die benötigten Ressourcen zu erhalten. Dies führt zu einer Verklemmung (deadlock).

8/8 ✓

3 Verklemmungsvermeidung (10 Punkte)

Sofern Ihre Implementierung der Aufgabe 1 zu einer Verklemmung führen kann, erweitern Sie Ihre Lösung um eine Verklemmungsvermeidung (deadlock avoidance), die Verklemmungen wirksam verhindert.

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

#define MAX 1000
// we assume that there are no more than 1000 philosophers on the same table.

int NUM_THREADS;
// how many philosophers

pthread_mutex_t chops[MAX];
// number of philosopher = number of chopstick,
// the rest of them are not available
// because of (right hand chopstick) = (i + 1) % Number of philosophers.


void *philosopher (void *threadid)
{
 long tid;
 tid = (long) threadid;
 int left = tid;
```

```
 int right = (tid + 1) % NUM_THREADS;

 while (1) {
  printf("philosopher %ld is thinking.\n", tid);

  printf("philosopher %ld is hungry and going to eat.\n", tid);


  if (tid % 2 == 0) {
  //Even-numbered philosophers take the chopsticks on the right-hand side first
  printf("philosopher %ld get chopsticks %d.\n", tid, right);
  sleep(1);
  printf("philosopher %ld get chopsticks %d.\n", tid, left);

  printf("philosopher %ld is eating.\n", tid);

  pthread_mutex_unlock(&chops[right]);
  sleep(1);
  pthread_mutex_unlock(&chops[left]);
  }
  else {
  //Odd-numbered philosophers take the chopsticks on the left-hand side first
  printf("philosopher %ld get chopsticks %d.\n", tid, left);
  sleep(1);
  printf("philosopher %ld get chopsticks %d.\n", tid, right);

  printf("philosopher %ld is eating.\n", tid);

  pthread_mutex_unlock(&chops[left]);
  sleep(1);
  pthread_mutex_unlock(&chops[right]);
  }
 }
  pthread_exit (NULL);
}


int main (/*int argc, char *argv[]*/)
{
 printf( "How many philosophers? :");
    scanf("%d", &NUM_THREADS);

  pthread_t threads[NUM_THREADS];
  int rc;
  long t;

  // init lock
  pthread_mutex_init(&chops[NUM_THREADS], NULL);

  // creating threads
  for (t=0; t < NUM_THREADS; t++) {
   rc = pthread_create (&threads[t], NULL, philosopher, (void *)t);
   if (rc) {
    printf ("ERROR; return code from pthread_create () is %d\n", rc);
    exit (-1);
   }
  }

  // joining threads
   for (long t = 0; t < NUM_THREADS; t++) {
    pthread_join (threads[t], NULL);
   }

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

Abbildung 2: Use 5 philosophers as an example

ich würde es eher als Prevention einordnen

Verklemmungsvermeidung (deadlock avoidance):
Es ist vorgesehen, dass ungeradzahlige Philosophen zuerst die linken und dann die rechten Essstäbchen nehmen, während die geradzahligen Philosophen das Gegenteil tun.

Wie im Diagramm dargestellt, werden die Philosophen 2,3 um die Stäbchen 3 konkurrieren und die Philosophen 4,5 um die Stäbchen 5 konkurrieren. Philosoph 1 muss nicht konkurrieren. Am Ende wird ein Philosoph immer zwei Stäbchen bekommen und essen.

Gute Idee!

8/10

Literatur

[1] ?

28/30