```
/*****************************************************************************/
/* Title: Uebung 5                                                           */
/* Description: Synchronisation zwischen parallel ablaufenden Threads        */
/*                                                                           */
/* Creator:                                                                  */
/* Matr.No: s721011 s782688                                                  */
/* Time of creation:                                                         */
/* Time of modification:                                                     */
/* Compile options: gcc -Wall u5.c -o u5                                     */
/*****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <errno.h>
#include <string.h>

#define NLOOP 10

int counter= 0;
/* shared variable */
pthread_mutex_t mutex= PTHREAD_MUTEX_INITIALIZER;

void dawdle(int cnt) {
  int i;
  double dummy= 3.141;
  for(i=0;i<cnt*10;i++) {
    dummy= dummy*dummy/dummy;
  }
  sleep(1);
}

/*****************************************************************************/
void *thread(void *args) {
  int i,val;
  int arg = *((int*)args);

  for (i= 0; i<NLOOP; i++) {
    pthread_mutex_lock(&mutex);
    val= counter;
    dawdle(arg);
    printf("Thread %lu: %d\n",pthread_self(),val+1);
    counter= val+1;
    pthread_mutex_unlock(&mutex);
  }
  pthread_exit(NULL);
}

/*****************************************************************************/
int main(int argc,char **argv) {
  int arg1=1, arg2=0;

  /*counter = shmget(IPC_PRIVATE,sizeof(int),IPC_CREAT|IPC_EXCL|0666);
  if(counter<0){
        perror("Fehler beim erzeugen von Schared Memory \n");
```

```
      exit(1);
  }*/

  pthread_t tidA,tidB;

  if (pthread_create(&tidA,NULL,thread,(void *)&arg1) != 0) {
    fprintf (stderr, "Konnte Thread 1 nicht erzeugen\n");
    exit (EXIT_FAILURE);
  }
  if (pthread_create(&tidB,NULL,thread,(void *)&arg2) != 0) {
    fprintf (stderr, "Konnte Thread 2 nicht erzeugen\n");
    exit (EXIT_FAILURE);
  }

  pthread_join(tidA, NULL);              //warte auf ende des Prozesses
  pthread_join(tidB, NULL);
  pthread_mutex_destroy(&mutex);  //pthread_mutex_destroy() function shall destroy the mutex
                                  object referenced by mutex

  return 0;

}

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

/*******************************************************************************************/
/* Title: Uebung 5                                                                         */
/* Description: Synchronisation zwischen parallel ablaufenden Threads                      */
/*                                                                                         */
/* Creator:                                                                                */
/* Matr.No: s721011 s782688                                                                */
/* Time of creation:                                                                       */
/* Time of modification:                                                                   */
/* Compile options: gcc -Wall -lpthread monitor.c diningphilos.c -o diningphilose          */
/*******************************************************************************************/

/**
 * \file monitor.c
 * \brief monitor functions for diningphilos
 * \author repat, repat@repat.de
 *
 * \note All comments for doxygen
*/

#include "diningphilos.h"

long countEat[NPHILO] = {0};
long countHungry[NPHILO] = {0};
long countThink[NPHILO] = {0};

/**
 * \brief philosopher tries to get both sticks or waits for sticks to become
 * available(on HIS cond-var), then eats
 * \param philoID philosoher ID from thread creation
```

```
 * \return nothing
 */
void
get_sticks(int philoID)
{

    //lock and display states
    pthread_mutex_lock(&mutex);
    disp_philo_states();

    //when he tries to get the sticks he's obviously hungry
    philoStates[philoID] = HUNGRY;

    while(stickStates[LEFT(philoID)] == IN_USE
          || stickStates[RIGHT(philoID)] == IN_USE) {
        pthread_cond_wait(&cond[philoID], &mutex);   // warten, mind. einer von zwei Sticks ist
vergeben
    }

    // mark philosopher as eating and stick as in use  // zwei Sticks sind verfuegbar
    philoStates[philoID] = EAT;
    stickStates[LEFT(philoID)] = IN_USE;                // Sticks werden benutzt
    stickStates[RIGHT(philoID)] = IN_USE;

          //unlock the mutex
    pthread_mutex_unlock(&mutex);
}

/**
 * \brief philosopher puts down the sticks, nudges his fellow philo-buddies and
 * enters thinking phase again
 * \param philoID philosoher ID from thread creation
 * \return nothing
 */
void
put_sticks(int philoID)
{

    // again, lock and display states
    pthread_mutex_lock(&mutex);
    disp_philo_states();

    // lets go of his sticks
    stickStates[LEFT(philoID)] = FREE;
    stickStates[RIGHT(philoID)] = FREE;

    // goes back to thinking
    philoStates[philoID] = THINK;

    // nudges his philosopher buddies next to him
    pthread_cond_signal(&cond[LEFTNEIGHB(philoID)]);
    pthread_cond_signal(&cond[RIGHTNEIGHB(philoID)]);

    pthread_mutex_unlock(&mutex);
}
```

```
/**
 * \brief Displays what happens inside the monitor like this
 * OT 1H 2E 3T 4T
 * in which T stands for THINK, H for HUNGRY and E for EAT
 * \return nothing
 */
void
disp_philo_states()
{
    int i;

    // go through them and look if there's an error
    for(i = 0; i < NPHILO; i++) {
        if(philoStates[i] == EAT
           && (philoStates[LEFTNEIGHB(i)] == EAT
               ll philoStates[RIGHTNEIGHB(i)] == EAT)) {
            printf("OUPS! Something went wrong...\n\n");
            break;
        }
    }

    // display anyway to see what might be wrong or actual states
    for(i = 0; i < NPHILO; i++)
    {
            switch (convertStates(philoStates[i]))
            {

                case 'E':
                        countEat[i]++;
                        break;
                case 'T':
                        countThink[i]++;
                        break;
                case 'H':
                        countHungry[i]++;
                        break;
            }

            printf("Philo:%d Eat:%ld Hungy:%ld Think:%ld \n", i, countEat[i], countHungry[i],
countThink[i]);
    }

    printf("\n\n");
}

/**
 * \brief Converts the states into their first letter
 * \param philoState state the philosoher is in(THINK, HUNGRY or EAT)
 * \return T/H/E for THINK, HUNGRY or EAT
 */
char
convertStates(State philoState)
{
    if(philoState == EAT) {
        return 'E';
    }
```

```
    else if(philoState == THINK) {
        return 'T';
    }
    else {
        return 'H';
    }
    return '-'; // error
}
```