BERGISCHE UNIVERSITÄT
GESAMTHOCHSCHULE WUPPERTAL
GAUSS-STRASSE 20
42097 WUPPERTAL
(Korrespondenzanschrift)
42119 WUPPERTAL
(Lieferanschrift)
TELEX 8 592 262 bughw
TELEFAX (0202) 439-2901
TELEFON (0202) 439-1

Fachbereich 7
MATHEMATIK

Prof. Dr. Hans-Jürgen Buhl
Praktische Informatik / Numerik

e-mail: Juergen.Buhl@math.uni-wuppertal.de

Betriebssysteme: Konzepte, Dienste, Schnittstellen (Betriebssysteme und betriebssystemnahe Programmierung)

SS 2003 – Übungsblatt 10

23. Juli 2003
Ausgabe: 16. Juli 2003

Aufgabe 1. *create and join threads*

Bringen Sie die folgenden Programme zum Ablauf

```
#include <stdio.h>
#include <pthread.h>

void print_message_function( void *ptr );

main(){
           pthread_t thread1, thread2;
           char *message1 = "Thread 1";
           char *message2 = "Thread 2";
           int  iret1, iret2;

    /* Create independant threads each of which will execute function */

           iret1 = pthread_create( &thread1, NULL,
                   (void*)&print_message_function, (void*) message1);
           iret2 = pthread_create( &thread2, NULL,
                   (void*)&print_message_function, (void*) message2);

   /* Wait till threads are complete before main continues. Unless we  */
   /* wait we run the risk of executing an exit which will terminate   */
   /* the process and all threads before the threads have completed.   */

           pthread_join( thread1, NULL);
           pthread_join( thread2, NULL);

           printf("Thread 1 returns: %d\n",iret1);
           printf("Thread 2 returns: %d\n",iret2);
           exit(0);
}

void print_message_function( void *ptr ){
           char *message;
           message = (char *) ptr;
           printf("%s \n", message);
}
```

und erklären Sie ihre jeweilige Wirkungsweise Zeile für Zeile.

Aufgabe 2. *mutex for synchronization*

Bringen Sie die folgenden Programme zum Ablauf

```
#include <stdio.h>
#include <pthread.h>

      void *functionC();
      pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
      int  counter = 0;

      main()
      {
         int rc1, rc2;
         pthread_t thread1, thread2;

   /* Create independant threads each of which will execute functionC */

         if( (rc1=pthread_create( &thread1, NULL, &functionC, NULL)) )
         {
            printf("Thread creation failed: %d\n", rc1);
         }

         if( (rc2=pthread_create( &thread2, NULL, &functionC, NULL)) )
         {
            printf("Thread creation failed: %d\n", rc2);
         }

   /* Wait till threads are complete before main continues. Unless we  */
   /* wait we run the risk of executing an exit which will terminate   */
   /* the process and all threads before the threads have completed.   */

         pthread_join( thread1, NULL);
         pthread_join( thread2, NULL);

         exit(0);
      }
      void *functionC()
      {
         pthread_mutex_lock( &mutex1 );
         counter++;
         printf("Counter value: %d\n",counter);
         pthread_mutex_unlock( &mutex1 );
      }
```

und erklären Sie ihre jeweilige Wirkungsweise Zeile für Zeile.

Aufgabe 3. *wait for 10 threads*

Bringen Sie das folgende Programm zum Ablauf

```
#include <stdio.h>
#include <pthread.h>

      #define NTHREADS 10
      void *thread_function();
      pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
      int  counter = 0;

      main()
      {
         pthread_t thread_id[NTHREADS];
         int i, j;

         for(i=0; i < NTHREADS; i++)
         {
            pthread_create( &thread_id[i], NULL, &thread_function, NULL );
         }

         for(j=0; j < NTHREADS; j++)
         {
            pthread_join( thread_id[j], NULL);
         }

 /* Now that all threads are complete I can print the final result.     */
 /* Without the join I could be printing a value before all the threads */
 /* have been completed.                                                 */

         printf("Final counter value: %d\n", counter);
      }

      void *thread_function()
      {
         printf("Thread number %ld\n", pthread_self());
         pthread_mutex_lock( &mutex1 );
         counter++;
         pthread_mutex_unlock( &mutex1 );
      }
```

und erklären Sie seine Wirkungsweise Zeile für Zeile.

Aufgabe 4. *conditional waiting*

Bringen Sie das folgende Programm zum Ablauf

```
#include <stdio.h>
#include <pthread.h>

      pthread_mutex_t count_mutex     = PTHREAD_MUTEX_INITIALIZER;
      pthread_mutex_t condition_mutex = PTHREAD_MUTEX_INITIALIZER;
      pthread_cond_t  condition_cond  = PTHREAD_COND_INITIALIZER;

      void *functionCount1();
      void *functionCount2();
      int  count = 0;
      #define COUNT_DONE  10
      #define COUNT_HALT1  3
      #define COUNT_HALT2  6

      main()
      {
         pthread_t thread1, thread2;

         pthread_create( &thread1, NULL, &functionCount1, NULL);
         pthread_create( &thread2, NULL, &functionCount2, NULL);
         pthread_join( thread1, NULL);
         pthread_join( thread2, NULL);

         exit(0);
      }

      void *functionCount1()
      {
         for(;;)
         {
            pthread_mutex_lock( &condition_mutex );
            while( count >= COUNT_HALT1 && count <= COUNT_HALT2 )
            {
               pthread_cond_wait( &condition_cond, &condition_mutex );
            }
            pthread_mutex_unlock( &condition_mutex );

            pthread_mutex_lock( &count_mutex );
            count++;
            printf("Counter value functionCount1: %d\n",count);
            pthread_mutex_unlock( &count_mutex );
            if(count >= COUNT_DONE) return(NULL);
          }
      }


      void *functionCount2()
      {
          for(;;)
          {
             pthread_mutex_lock( &condition_mutex );
             if( count < COUNT_HALT1 || count > COUNT_HALT2 )
             {
                pthread_cond_signal( &condition_cond );
             }
             pthread_mutex_unlock( &condition_mutex );

             pthread_mutex_lock( &count_mutex );
             count++;
             printf("Counter value functionCount2: %d\n",count);
             pthread_mutex_unlock( &count_mutex );

             if(count >= COUNT_DONE) return(NULL);
          }

      }
```

und erklären Sie seine Wirkungsweise Zeile für Zeile.

Aufgabe 5. *Variation der Anzahl von Threads*

Bringen Sie das folgende Programm zum Ablauf

```
/****************************************************/
/* Another thread example.  This one shows that    */
/* pthreads in Linux can use both processors in    */
/* a dual-processor Pentium.                        */
/*                                                  */
/* Usage:  a.out <num threads>                      */
/*                                                  */
/* To compile me in Linux type:                     */
/*    gcc -o another another.c -lpthread            */
/****************************************************/

#include <pthread.h>
#include <stdio.h>
#include <math.h>

#define MAX_THREADS 10
#define UPPER_LIM 8000000
```

```
int last=1;

int sum; /* this data is shared by the thread(s) */
void *runner(void * param);

main(int argc, char *argv[])
{
  int num_threads, i;
  pthread_t tid[MAX_THREADS];     /* the thread identifiers  */
  pthread_attr_t attr; /* set of thread attributes */

  if (argc != 2) {
    fprintf(stderr, "usage:  pthread5 <integer value>\n");
    exit(0);
  }

  if (atoi(argv[1]) <= 0) {
    fprintf(stderr,"%d must be > 0\n", atoi(argv[1]));
    exit(0);
  }

  if (atoi(argv[1]) > MAX_THREADS) {
    fprintf(stderr,"%d must be <= %d\n", atoi(argv[1]), MAX_THREADS);
    exit(0);
  }

  num_threads = atoi(argv[1]);
  printf("The number of threads is %d\n", num_threads);

  last = UPPER_LIM / atoi(argv[1]);

  /* get the default attributes */
  pthread_attr_init(&attr);

  /* create the threads */
  for (i=0; i<num_threads; i++) {
    pthread_create(&(tid[i]), &attr, runner, (void *) i);
    printf("Creating thread number %d, tid=%lu \n", i, tid[i]);
  }

  /* now wait for the threads to exit */
  for (i=0; i<num_threads; i++) {
    pthread_join(tid[i],NULL);
  }

  for (i=0; i<num_threads; i++) {
     printf("last = %d in thread %d \n", last, i);
  }
}

/* The thread will begin control in this function */
void *runner(void * param)
{
  int i;
  int j;
  int threadnumber = (int) param;
  for (i=0; i<last; i++){
    j = sin(i*i);
    /* printf("Thread number=%d, j=%d\n", threadnumber, j); */
  }
  pthread_exit(0);
}
```

und erklären Sie seine Wirkungsweise Zeile für Zeile.