answers/Code_Q3_4/Q3/exampleSyncUpdated2/deadlock.c

```c
/*
 * Author: Sam Siewert
 * Modified by: Shashank and Parth
 * Description: Added random backoff scheme to avoid deadlock
 */

#include <pthread.h>
#include <stdio.h>
#include <sched.h>
#include <time.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#define NUM_THREADS 2
#define THREAD_1 0
#define THREAD_2 1

typedef struct
{
    int threadIdx;
} threadParams_t;


pthread_t threads[NUM_THREADS];
threadParams_t threadParams[NUM_THREADS];

struct sched_param nrt_param;

// On the Raspberry Pi, the MUTEX semaphores must be statically initialized
//
// This works on all Linux platforms, but dynamic initialization does not work
// on the R-Pi in particular as of June 2020.
//
pthread_mutex_t rsrcA = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t rsrcB = PTHREAD_MUTEX_INITIALIZER;

volatile int rsrcACnt=0, rsrcBCnt=0, noWait=0, backoff=0;

/********************Random back off scheme to avoid deadlock*******************/
int random_backoff_scheme(void)
{
  int random_backoff_time;
  random_backoff_time = (rand() % 3) + 2;   // Generating delay between 2 to 5
seconds (2 added as minimum delay needed is 2 to avoid deadlock)
  return random_backoff_time;
}


void *grabRsrcs(void *threadp)
{
   threadParams_t *threadParams = (threadParams_t *)threadp;
   int threadIdx = threadParams->threadIdx;
```

```c


    if(threadIdx == THREAD_1)
    {
      printf("THREAD 1 grabbing resources\n");
      pthread_mutex_lock(&rsrcA);
      rsrcACnt++;
      if(!noWait) sleep(1);
      printf("THREAD 1 got A, trying for B\n");
      pthread_mutex_lock(&rsrcB);
      rsrcBCnt++;
      printf("THREAD 1 got A and B\n");
      pthread_mutex_unlock(&rsrcB);
      pthread_mutex_unlock(&rsrcA);
      printf("THREAD 1 done\n");
    }
    else
    {
      //Random backoff delay for thread 2 so that thread 1 can acquire the mutex rsrcB and finish execution
      if(backoff)
      {
        int random_backoff_delay = random_backoff_scheme();
        printf("Random backoff time is %d seconds\n",random_backoff_delay);
        sleep(random_backoff_delay);
      }
      printf("THREAD 2 grabbing resources\n");
      pthread_mutex_lock(&rsrcB);
      rsrcBCnt++;
      if(!noWait) sleep(1);
      printf("THREAD 2 got B, trying for A\n");
      pthread_mutex_lock(&rsrcA);
      rsrcACnt++;
      printf("THREAD 2 got B and A\n");
      pthread_mutex_unlock(&rsrcA);
      pthread_mutex_unlock(&rsrcB);
      printf("THREAD 2 done\n");
    }
    pthread_exit(NULL);
}


int main (int argc, char *argv[])
{
    int rc, safe=0;

    rsrcACnt=0, rsrcBCnt=0, noWait=0, backoff=0;

    srand(time(NULL)); //Initialize random number generator

    if(argc < 2)
    {
      printf("Will set up unsafe deadlock scenario\n");
    }
    else if(argc == 2)
    {
```

```c
    if(strncmp("safe", argv[1], 4) == 0)
      safe=1;
    else if(strncmp("race", argv[1], 4) == 0)
      noWait=1;
    else if(strncmp("backoff", argv[1], 7) == 0)
      backoff=1;
    else
      printf("Will set up unsafe deadlock scenario\n");
  }
  else
  {
    printf("Usage: deadlock [safe|race|unsafe]\n");
  }


  printf("Creating thread %d\n", THREAD_1+1);
  threadParams[THREAD_1].threadIdx=THREAD_1;
  rc = pthread_create(&threads[0], NULL, grabRsrcs, (void *)&threadParams[THREAD_1]);
  if (rc) {printf("ERROR; pthread_create() rc is %d\n", rc); perror(NULL); exit(-1);}
  printf("Thread 1 spawned\n");

  if(safe) // Make sure Thread 1 finishes with both resources first
  {
    if(pthread_join(threads[0], NULL) == 0)
      printf("Thread 1: %x done\n", (unsigned int)threads[0]);
    else
      perror("Thread 1");
  }

  printf("Creating thread %d\n", THREAD_2+1);
  threadParams[THREAD_2].threadIdx=THREAD_2;
  rc = pthread_create(&threads[1], NULL, grabRsrcs, (void *)&threadParams[THREAD_2]);
  if (rc) {printf("ERROR; pthread_create() rc is %d\n", rc); perror(NULL); exit(-1);}
  printf("Thread 2 spawned\n");

  printf("rsrcACnt=%d, rsrcBCnt=%d\n", rsrcACnt, rsrcBCnt);
  printf("will try to join CS threads unless they deadlock\n");

  if(!safe)
  {
    if(pthread_join(threads[0], NULL) == 0)
      printf("Thread 1: %x done\n", (unsigned int)threads[0]);
    else
      perror("Thread 1");
  }

  if(pthread_join(threads[1], NULL) == 0)
    printf("Thread 2: %x done\n", (unsigned int)threads[1]);
  else
    perror("Thread 2");

  if(pthread_mutex_destroy(&rsrcA) != 0)
    perror("mutex A destroy");

    if(pthread_mutex_destroy(&rsrcB) != 0)
      perror("mutex B destroy");

    printf("All done\n");

    exit(0);
}

```