```
/*
 * Demonstrate deadlock avoidance using "mutex backoff".
 *
 * Special notes: On a Solaris 2.5 uniprocessor, this test will
 * not produce interleaved output unless extra LWPs are created
 * by calling thr_setconcurrency(), because threads are not
 * timesliced.
 */
#include <pthread.h>
#include <sched.h>
#include "errors.h"

#define ITERATIONS 10

pthread_mutex_t mutex[3];
int backoff = 1;        /* Whether to backoff or deadlock */
int yield_flag = 0;     /* 0: no yield, >0: yield, <0: sleep */

/*
 * This is a thread start routine that locks all mutexes in
 * order, to ensure a conflict with lock_reverse, which does the
 * opposite.
 */

void *lock_forward (void *arg) {
  int i, j, iterate, backoffs;
  int status;

  for (iterate = 0; iterate<ITERATIONS; iterate++) {
    backoffs = 0;
    for (i=0; i<3; i++) {
      if (i==0) {
        status = pthread_mutex_lock (&mutex[i]);
        sleep(1);
        if (status != 0)
          err_abort (status, "First lock");
      } else {
        if (backoff)
          status = pthread_mutex_trylock (&mutex[i]);
        else
          status = pthread_mutex_lock (&mutex[i]);
        if (status == EBUSY) {
          backoffs++;
          for (j=0; j<i; j++) printf ("\t");
          printf (" [forward locker backing off at %d]\n", i);
          for (; i>=0; i--) {
            status = pthread_mutex_unlock (&mutex[i]);
              if (status != 0)
                err_abort (status, "Backoff");
            }
        } else {
          if (status != 0)
            err_abort (status, "Lock mutex");
          for (j=0; j<i; j++) printf ("\t");
          printf (" [forward locker got %d]\n", i);
        }
      }
      /*
       * Yield processor, if needed to be sure locks get
       * interleaved on a uniprocessor.
```

```
       */
      if (yield_flag) {
        if (yield_flag > 0)
          sched_yield ();
        else
          sleep (1);
      }
    }
    /*
     * Report that we got 'em, and unlock to try again.
     */
    printf (
        "lock forward got all locks, %d backoffs\n", backoffs);
    pthread_mutex_unlock (&mutex[2]);
    pthread_mutex_unlock (&mutex[1]);
    pthread_mutex_unlock (&mutex[0]);
    sched_yield ();
  }
  return NULL;
}

/*
 * This is a thread start routine that locks all mutexes in
 * reverse order, to ensure a conflict with lock_forward, which
 * does the opposite.
 */
void *lock_backward (void *arg) {
  int i, j, iterate, backoffs;
  int status;

  for (iterate=0; iterate<ITERATIONS; iterate++) {
    backoffs = 0;
    for (i=2; i>=0; i--) {
      if (i==2) {
        status = pthread_mutex_lock (&mutex[i]);
        sleep(1);
        if (status != 0)
          err_abort (status, "First lock");
      } else {
        if (backoff)
          status = pthread_mutex_trylock (&mutex[i]);
        else
          status = pthread_mutex_lock (&mutex[i]);
        if (status == EBUSY) {
          backoffs++;
          for (j=0; j<i; j++) printf ("\t");
          printf (" [backward locker backing off at %d]\n", i);
          for (; i < 3; i++) {
            status = pthread_mutex_unlock (&mutex[i]);
            if (status != 0)
              err_abort (status, "Backoff");
            }
        } else {
          if (status != 0)
            err_abort (status, "Lock mutex");
          for (j=0; j<i; j++) printf ("\t");
          printf (" [backward locker got %d]\n", i);
        }
      }
      /*
```

```
        * Yield processor, if needed to be sure locks get
        * interleaved on a uniprocessor.
        */
       if (yield_flag) {
         if (yield_flag > 0)
           sched_yield ();
         else
           sleep (1);
       }
     }
     /*
      * Report that we got 'em, and unlock to try again.
      */
     printf (
         "lock backward got all locks, %d backoffs\n", backoffs);
     pthread_mutex_unlock (&mutex[0]);
     pthread_mutex_unlock (&mutex[1]);
     pthread_mutex_unlock (&mutex[2]);
     sched_yield ();
   }
   return NULL;
}

int main (
   int argc,
   char *argv[]
   )
{
   pthread_t forward, backward;
   int i, status;

#ifdef sun
   /*
    * On Solaris 2.5, threads are not timesliced. To ensure
    * that our threads can run concurrently, we need to
    * increase the concurrency level.
    */
   printf ("Setting concurrency level to 2\n");
   thr_setconcurrency (2);
#endif

   /*
    * If the first argument is absent, or nonzero, a backoff
    * algorithm will be used to avoid deadlock. If the first
    * argument is zero, the program will deadlock on a lock
    * "collision."
    */
   if (argc > 1) {
     backoff = atoi (argv[1]);
   }

   /*
    * If the second argument is absent, or zero, the two
    * threads run "at speed." On some systems, especially
    * uniprocessors, one thread may complete before the other
    * has a chance to run, and you won't see a deadlock or
    * backoffs. In that case, try running with the argument set
    * to a positive number to cause the threads to call
    * sched_yield() at each lock; or, to make it even more
    * obvious, set to a negative number to cause the threads to
```

```
   * call sleep(1) instead.
   */
  if (argc > 2) {
    yield_flag = atoi (argv[2]);
  }

  for (i=0; i<3; i++) {
    pthread_mutex_init (&mutex[i], NULL);
  }

  status = pthread_create (&forward, NULL, lock_forward, NULL);
  if (status != 0)
    err_abort (status, "Create forward");

  status = pthread_create (&backward, NULL, lock_backward, NULL);
  if (status != 0)
    err_abort (status, "Create backward");

  pthread_exit (NULL);
}
```