Pseudo code for the pthread based donut problem

PRODUCER

```
get prod mutex
check space count
loop:
  if space count == 0
     wait prod_condx_var
put donut in queue
decrement space counter
increment serial number, in ptr
unlock prod mutex

get cons mutex
inc donut count
unlock cons mutex
signal cons_condx_var
```

CONSUMER

```
get cons mutex
check donut count
loop:
  if donut count == 0
     wait cons_condx_var
take donut from queue
decrement donut counter
increment out ptr
unlock cons mutex

get prod mutex
inc space count
unlock prod mutex
signal prod_condx_var
```

Remember, when a condx_wait is called the associated mutex is implicitly released by the system and when the wait returns the system guarantees that the associated mutex has been re-acquired for the waking thread and that it is "safe" to proceed.

THE FOLLOWING CODE EXAMPLES SHOULD PROVIDE HELP WITH THE pthread IMPLEMENTATION OF THE DONUTS PROBLEM....THIS VERSION INCLUDES A SIGNAL MANAGEMENT THREAD WHICH RESPONDS TO A SIGTERM (signal #15) SIGNAL....I INCLUDED IT AS A WAY OF STOPPING A RUN WHICH GETS INTO A DEADLOCK....JUST START YOUR PROGRAM IN THE BACKGROUND, AND IF IT STOPS MAKING PROGRESS SEND IT SIGTERM FROM THE KEYBOARD (i.e. shell prompt>> kill PID#)....THIS USE OF KILL WILL SEND SIGTERM BY DEFAULT....THE PROGRAM ALSO HAS TIME STAMP PROCEDURES WHICH COLLECT INFORMATION ABOUT HOW LONG (wall clock, not execution time) IT TOOK A RUN TO COMPLETE....YOU CAN USE THE SIGNAL CODE VERBATIM (I will discuss signal management with threads in class)

compile line:

```
           gcc -o my_th_donuts my_th_donuts.c  -lpthread
```

```
/* INCLUDE FILE STUFF, THESE BELONG IN A  .h   FILE             */
/*****************************************************************/
#include <unistd.h>
#include <signal.h>
#include <sys/time.h>
#include <pthread.h>
#define         NUMFLAVORS          4
#define         NUMSLOTS            50
#define         NUMCONSUMERS        5
#define         NUMPRODUCERS        5
typedef struct {
       int            flavor [NUMFLAVORS] [NUMSLOTS];
       int            outptr [NUMFLAVORS];
       int            in_ptr [NUMFLAVORS];
       int            serial [NUMFLAVORS];
       int            spaces [NUMFLAVORS];
       int            donuts [NUMFLAVORS];
} DONUT_SHOP;
/*****************************************************************/
/* SIGNAL WAITER, PRODUCER AND CONSUMER THREAD FUNCTIONS         */
/*****************************************************************/
       void   *sig_waiter ( void *arg );
       void   *producer   ( void *arg );
       void   *consumer   ( void *arg );
       void    sig_handler ( int );
```

```
             /****************************/
             /*      GLOBAL VARIABLES    */
             /****************************/

#include "project_header.h"

DONUT_SHOP          shared_ring;

pthread_mutex_t     prod [NUMFLAVORS];
pthread_mutex_t     cons [NUMFLAVORS];
pthread_cond_t      prod_cond [NUMFLAVORS];
pthread_cond_t      cons_cond [NUMFLAVORS];
pthread_t           thread_id [NUMCONSUMERS+NUMPRODUCERS];
pthread_t           sig_wait_id;
```

```
int  main ( int argc, char *argv[] )
{
       int                 i, j, k, nsigs;
       struct timeval      randtime, first_time, last_time;
       struct sigaction    new_act;
       int                 arg_array[NUMCONSUMERS];
       sigset_t            all_signals;
       int sigs[]          = { SIGBUS, SIGSEGV, SIGFPE };

       pthread_attr_t      thread_attr;
       struct sched_param  sched_struct;

/***********************************************************/
/* INITIAL TIMESTAMP VALUE FOR PERFORMANCE MEASURE         */
/***********************************************************/

        gettimeofday (&first_time, (struct timezone *) 0 );

        /******** SET ARRAY OF ARGUMENT VALUES *********/
        for ( i = 0; i < NUMCONSUMERS ; i++ ) {
             arg_array [i] = i + 1;   /* cons[0] has ID = 1 */
        }
```

```
/************************************************************/
/* GENERAL PTHREAD MUTEX AND CONDITION INIT AND GLOBAL INIT */
/************************************************************/

       for ( i = 0; i < NUMFLAVORS; i++ ) {
              pthread_mutex_init ( &prod [i], NULL );
              pthread_mutex_init ( &cons [i], NULL );
              pthread_cond_init  ( &prod_cond [i],  NULL );
              pthread_cond_init  ( &cons_cond [i],  NULL );
              shared_ring.outptr [i]           = 0;
              shared_ring.in_ptr [i]           = 0;
              shared_ring.serial [i]           = 0;
              shared_ring.spaces [i]           = NUMSLOTS;
              shared_ring.donuts [i]           = 0;
       }
```

```
/*************************************************************/
/* SETUP FOR MANAGING THE SIGTERM SIGNAL, BLOCK ALL SIGNALS */
/*************************************************************/

       sigfillset (&all_signals );
       nsigs = sizeof ( sigs ) / sizeof ( int )
       for ( i = 0; i < nsigs; i++ )
              sigdelset ( &all_signals, sigs [i] );

       sigprocmask ( SIG_BLOCK, &all_signals, NULL );
       sigfillset (&all_signals );
       for( i = 0; i <  nsigs; i++ ) {
              new_act.sa_handler  = sig_handler;
              new_act.sa_mask            = all_signals;
              new_act.sa_flags           = 0;
              if ( sigaction (sigs[i], &new_act, NULL) == -1 ){
                            perror("can't set signals: ");
                            exit(1);
              }
       }
       printf ( "just before threads created\n" );
```

```
/********************************************************/
/* CREATE SIGNAL HANDLER THREAD, PRODUCER AND CONSUMERS */
/********************************************************/
        if ( pthread_create (&sig_wait_id, NULL,
                                   sig_waiter, NULL) != 0 ){
                printf ( "pthread_create failed " );
                exit ( 3 );
        }

       pthread_attr_init            ( &thread_attr );
       pthread_attr_setinheritsched ( &thread_attr,
                                      PTHREAD_INHERIT_SCHED );
   #ifdef  GLOBAL
        pthread_attr_setinheritsched ( &thread_attr,
                                      PTHREAD_EXPLICIT_SCHED );
        pthread_attr_setschedpolicy ( &thread_attr, SCHED_OTHER );

        sched_struct.sched_priority =
                               sched_get_priority_max(SCHED_OTHER);
        pthread_attr_setschedparam ( &thread_attr, &sched_struct );

        pthread_attr_setscope      ( &thread_attr,
                                            PTHREAD_SCOPE_SYSTEM );
   #endif
```

```
for ( i = 0; i < NUMCONSUMERS ; i++, j++ ) {
       if ( pthread_create ( &thread_id [i], &thread_attr,
                 consumer, ( void * )&arg_array [i]) != 0 ){
             printf ( "pthread_create failed" );
             exit ( 3 );
       }
}

for ( ; i < NUMPRODUCERS + NUMCONSUMERS; i++ ) {
       if ( pthread_create (&thread_id[i], &thread_attr,
                                   producer, NULL ) != 0 ) {
             printf ( "pthread_create failed " );
             exit ( 3 );
       }
}
printf ( "just after threads created\n" );
```

```
/********************************************************************/
/* WAIT FOR ALL CONSUMERS TO FINISH, SIGNAL WAITER WILL            */
/* NOT FINISH UNLESS A SIGTERM ARRIVES AND WILL THEN EXIT          */
/* THE ENTIRE PROCESS....OTHERWISE MAIN THREAD WILL EXIT           */
/* THE PROCESS WHEN ALL CONSUMERS ARE FINISHED                     */
/********************************************************************/
        for ( i = 0; i < NUMCONSUMERS; i++ )
                      pthread_join ( thread_id [i], NULL );
/*****************************************************************/
/* GET FINAL TIMESTAMP, CALCULATE ELAPSED SEC AND USEC          */
/*****************************************************************/
        gettimeofday (&last_time, ( struct timezone * ) 0 );
        if ( ( i = last_time.tv_sec - first_time.tv_sec) == 0 )
               j = last_time.tv_usec - first_time.tv_usec;
        else{
               if ( last_time.tv_usec - first_time.tv_usec < 0 ) {
                      i--;
                      j = 1000000 +
                          ( last_time.tv_usec - first_time.tv_usec );
               } else {
                      j = last_time.tv_usec - first_time.tv_usec; }
       }
        printf ( "Elapsed cons time is %d sec and %d usec\n", i, j );

        printf ( "\n\n ALL CONSUMERS FINISHED, KILLING  PROCESS\n\n" );
        exit ( 0 );
}
```

```
        /*********************************************/
        /*      INITIAL PART OF PRODUCER.....        */
        /*********************************************/
void    *producer ( void *arg )
{
        int                   i, j, k;
        unsigned short        xsub [3];
        struct timeval        randtime;
        gettimeofday ( &randtime, ( struct timezone * ) 0 );
        xsub1 [0] = ( ushort ) randtime.tv_usec;
        xsub1 [1] = ( ushort ) ( randtime.tv_usec >> 16 );
        xsub1 [2] = ( ushort ) ( pthread_self );
        while ( 1 ) {
          j = nrand48 ( xsub ) & 3;
          pthread_mutex_lock ( &prod [j] );
           while ( shared_ring.spaces [j] == 0 ) {
                pthread_cond_wait ( &prod_cond [j], &prod [j] );
          }
               .  /* safe to manipulate in_ptr, serial      */
               .  /* counter and space counter for flavor j */
          pthread_mutex_unlock ( &prod [j] );
               .  /* now need to increase j donut count, etc.*/
               .  /* but this will require another mutex . . */
        return NULL;
} /* end main */
```

```
        /*******************************************/
        /*    ON YOUR OWN FOR THE CONSUMER........ */
        /*******************************************/

  void     *consumer ( void *arg )
  {
         int                   i, j, k, m, id;
         unsigned short        xsub [3];
         struct timeval        randtime;
         id = *( int * ) arg;
         gettimeofday ( &randtime, ( struct timezone * ) 0 );
         xsub [0] = ( ushort ) randtime.tv_usec;
         xsub [1] = ( ushort ) ( randtime.tv_usec >> 16 );
         xsub [2] = ( ushort ) ( pthread_self );

          for( i = 0; i < 10; i++ ) {
            for( m = 0; m < 12; m++ ) {

                j = nrand48( xsub ) & 3;
                .
                ...etc.........
              } /* end getting 1 doz, now context switch? */
```

```
/******************************************************/
/*     THREAD YIELD WILL ALLOW ANOTHER CONSUMER       */
/* AFTER EACH DOZEN … ( COULD ALSO USE  usleep(100) ) */
/******************************************************/

       sched_yield ( );  /* for system scope threads  */
                    OR
       usleep ( 100 );   /* for process scope threads */


    } /* end getting 10 dozen */

    return NULL:

}     /* end main */
```

```
/***********************************************************/
/*         PTHREAD ASYNCH SIGNAL HANDLER ROUTINE...        */
/***********************************************************/

void    *sig_waiter ( void *arg ){

       sigset_t     sigterm_signal;
       int          signo;

       sigemptyset ( &sigterm_signal );
       sigaddset   ( &sigterm_signal, SIGTERM );
       sigaddset   ( &sigterm_signal, SIGINT );

   /* set for asynch signal management for SIGs 2 and 15  */

       if sigwait ( &sigterm_signal, & signo)  != 0 ) {
              printf ( "\n  sigwait ( ) failed, exiting \n");
              exit(2);
       }
       printf ( ”Process exits on SIGNAL %d\n\n", signo );
       exit ( 1 );
       return NULL;  /* not reachable */
}
```

```
/**********************************************************/
/*         PTHREAD SYNCH SIGNAL HANDLER ROUTINE...        */
/**********************************************************/

void   sig_handler ( int sig ){

       pthread_t    signaled_thread_id;
       int          i, thread_index;

       signaled_thread_id = pthread_self ( );

/*******  check for own ID in array of thread Ids *******/

       for ( i = 0; i < ( NUMCONSUMERS ); i++) {
              if ( signaled_thread_id == thread_id [i] )  {
                            thread_index = i + 1;
                            break;
              }
       }
       printf ( "\nThread %d took signal # %d, PROCESS HALT\n",
                            thread_index, sig );
       exit ( 1 );
}
```

```
/*** Checking the inherited process affinity mask ***/
#define _GNU_SOURCE
#include <sched.h>
#include <utmpx.h>

cpu_set_t      mask;

sched_getaffinity(syscall(SYS_gettid), sizeof(cpu_set_t), &mask);
for(i=0; i<24; ++i)proc_cnt += (CPU_ISSET(i, &mask))?1:0;

printf("\nPROCESS AFFINITY MASK BEFORE ADJUSTMENT:\n");
printf(" CPUs : 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 ....

printf(wr_buf, " %s: %d %d %d %d %d %d  ....%d  %d\n", "     ",
                      (CPU_ISSET(0, &mask))?1:0,
                      (CPU_ISSET(1, &mask))?1:0,
                      (CPU_ISSET(2, &mask))?1:0,
                      (CPU_ISSET(3, &mask))?1:0,
                      (CPU_ISSET(4, &mask))?1:0,
                      ....

write(1,wr_buf,strlen(wr_buf));
```

```
/* Setting and checking process affinity mask after setting */

CPU_ZERO(&mask);
proc_cntx =  (nrand48(xsub1));
CPU_SET(proc_cntx%proc_cnt, &mask);
sched_setaffinity(0, sizeof(cpu_set_t), &mask);

printf("\nPROCESS AFFINITY MASK AFTER ADJUSTMENT:\n");

sprintf(wr_buf, " %s: %d %d %d %d %d %d  ....%d  %d\n", "     ",
                      (CPU_ISSET(0, &mask))?1:0,
                      (CPU_ISSET(1, &mask))?1:0,
                      (CPU_ISSET(2, &mask))?1:0,
                      (CPU_ISSET(3, &mask))?1:0,
                      (CPU_ISSET(4, &mask))?1:0,
                      ....

write(1,wr_buf,strlen(wr_buf));
```

```
/*** Setting and checking individual thread affinity mask ***/

sched_getaffinity(syscall(SYS_gettid), sizeof(cpu_set_t), &mask);
for(i=0; i<24; ++i)proc_cnt += (CPU_ISSET(i, &mask))?1:0;

CPU_ZERO(&mask);
CPU_SET(my_id%proc_cnt, &mask);
sched_setaffinity(0, sizeof(cpu_set_t), &mask);

sched_getaffinity(syscall(SYS_gettid), sizeof(cpu_set_t), &mask);
sprintf(wr_buf, " %s: %d %d %d %d %d %d  ....%d  %d\n", "     ",
                      (CPU_ISSET(0, &mask))?1:0,
                      (CPU_ISSET(1, &mask))?1:0,
                      (CPU_ISSET(2, &mask))?1:0,
                      (CPU_ISSET(3, &mask))?1:0,
                      (CPU_ISSET(4, &mask))?1:0,
                      ....

write(1,wr_buf,strlen(wr_buf));
```