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 re-check the control variable for the value you need. If the control variable is still not the value that you need it to be, you should go back to sleep with another condx_wait call.

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 (this is used by the run script). THE PROGRAM ALSO HAS TIME STAMP PROCEDURES WHICH COLLECT INFORMATION ABOUT HOW LONG (wall clock, not execution time) IT TOOK A RUN TO COMPLETION.

compile line:

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

```
/* YOU WILL NEED THE FOLLOWING INCLUDE FILES  */
/**********************************************/

#define _GNU_SOURCE
#include <sched.h>
#include <utmpx.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/syscall.h>
#include <linux/unistd.h>
#include <strings.h>
#include <signal.h>
#include <sys/time.h>
#include <pthread.h>
#include <sys/fcntl.h>
#include <stdio.h>
#include <errno.h>
```

```
/* INCLUDE FILE STUFF, THESE BELONG IN A  .h   FILE              */
/*****************************************************************/
#include <unistd.h>
#include <signal.h>
//  etc. see last slide
#include <pthread.h>
#define         NUMFLAVORS            4
#define         NUMSLOTS             1900
#define         NUMCONSUMERS          50
#define         NUMPRODUCERS          30
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;
       unsigned int        cpu;
       int                 proc_cnt=0;
       int                 proc_cntx, cn;
       float               etime;
       ushort              xsub1[3];
       cpu_set_t           mask;
```

```
/***********************************************************/
/* 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("\n\nElapsed consumer time is %d sec and %d usec, or %f sec\n",
              i, j, (etime =i + (float)j/1000000));
    if((cn = open("./run_times", O_WRONLY|O_CREAT|O_APPEND, 0644)) == -1){
        perror("can not open sys time file ");
        exit(1);
    }
    sprintf(msg, "%f\n", etime);
    write(cn, msg, strlen(msg));
```

```
        /*********************************************/
        /*      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 producer
```

```
      /*********************************************/
      /*     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 one dozen, now sleep */
           usleep (100);
          } /* end getting 10 dozen */
} /* end consumer */
```

```
/***********************************************************/
/*          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 );
}
```

mercury.cs.uml.edu 10-9-2014

Producers	=	10
Consumers	=	35
Qdepth	=	2000
Cons dozns	=	2500
Donut flav	=	4
Thrd scope	=	Process
Numbr CPUs	=	8

System scope(us)		Process scope(us)	
9.4427		4.6857	
9.4587		4.9011	
9.3479		4.9249	
9.4832		4.9271	
9.4157		4.9163	
9.3863		4.8904	
9.4707		4.8839	
9.3337		4.8518	
9.4310		5.0109	
9.4336		4.7719	
9.3872		4.8776	
9.4173	μ	**4.8765**	μ
0.0486	σ	**0.0856**	σ