```
/*
 * catsem.c
 *
 * 30-1-2003 : GWA : Stub functions created for CS161 Asst1.
 *
 * NB: Please use SEMAPHORES to solve the cat syncronization problem in
 * this file.
 */


/*
 *
 * Includes
 *
 */

#include <types.h>
#include <lib.h>
#include <test.h>
#include <thread.h>


/*
 *
 * Constants
 *
 */

/*
 * Number of food bowls.
 */

#define NFOODBOWLS 2

/*
 * Number of cats.
 */

#define NCATS 6

/*
 * Number of mice.
 */

#define NMICE 2

/*
 * Declare cat/mouse semaphores
 */

static struct semaphore *bowlsem;
static struct semaphore *catsemaphore;
static struct semaphore *mousesemaphore;
static struct semaphore *printsem;

/*
 * Declare static volatile variables
 */

static volatile unsigned int cycle;
static volatile unsigned int catseating;
```

```
static volatile unsigned int miceeating;
static volatile unsigned int catsdoneeating;
static volatile unsigned int micedoneeating;
static volatile unsigned int bowllist[NFOODBOWLS];

/*
 *
 * Function Definitions
 *
 */


/*
 * catsem()
 *
 * Arguments:
 *      void * unusedpointer: currently unused.
 *      unsigned long catnumber: holds the cat identifier from 0 to
NCATS - 1.
 *
 * Returns:
 *      nothing.
 *
 * Notes:
 *      This function represents the simulation of a cat. Upon
 *execution each "cat" takes a resource of catsemaphore, which will
 *be initialized to either NCATS or 0. After this a cat will take
 *a bowl. Depending on the number of bowls available cats may wait for
 *bowls to be unoccupied before continuing. That being said, the cat
 *then eats, and replaces the bowl. When all cats are done, and if no
 *mice have eaten, we allocate NMICE resources to mousesemaphore so
 *mice can eat.
 *
 */

static void catsem(void * unusedpointer,
                           unsigned long catnumber)
{
      int i; //Declare loop variable
      int c;      //Assigns a cycle number to this thread
      int bowl;   //Stores current bowl in use

      //Avoid unused variable warnings.
      (void) unusedpointer;

      //Cats begin or stop
      P(catsemaphore);

      //Cat starts eating
      P(bowlsem); //Take one bowl

      catseating++;

      if(catseating == 1)
      {
            P(printsem);
            kprintf("Cats are starting to eat\n");
            V(printsem);
      }

      for(i = 0; i < NFOODBOWLS; i++) //Used to assign bowl #
```

```
	{
		if(bowllist[i] == 1)
		{
			bowl = i; //This cat takes a bowl
			bowllist[i] = 0; //Remove bowl from list
			break;
		}
	}
	P(printsem);
	c = cycle++;
	kprintf("Cycle %u: Cat %u beginning to eat at bowl %u\n", c,
catnumber, bowl+1);
	V(printsem);
	//Simulate Eating Time, no mice can be eating
	for(i = 0; i < 5000; i++)
		assert(miceeating == 0);
	//Place bowl back into list
	bowllist[bowl] = 1;
	P(printsem);
	kprintf("Cycle %u: Cat %u finishing eating at bowl %u\n", c,
catnumber, bowl+1);
	V(printsem);
	//Cat finished eating
	catsdoneeating++;
	catseating--;
	V(bowlsem); //Replace one bowl

	//True when last catsem thread executes this line of code
	if(catsdoneeating == NCATS)
	{
		//Catsem complete, all cats have finished eating
		P(printsem);
		kprintf("All %u cats have finished eating\n",
catsdoneeating);
		V(printsem);
		//Allow mice to eat if they havn't done so
		if(micedoneeating == 0)
		{
			for(i = 0; i < NMICE; i++)
				V(mousesemaphore);
		}
	}
}


/*
 * mousesem()
 *
 * Arguments:
 *	void * unusedpointer: currently unused.
 *	unsigned long mousenumber: holds the mouse identifier from 0 to
 *		NMICE - 1.
 *
 * Returns:
 *	nothing.
 *
 * Notes:
 *	Same as catsem, only reversed for mice.
 *
 */
```

```
static void mousesem(void * unusedpointer,
                                unsigned long mousenumber)
{
      int i;      //Declare loop variable
      int c;      //Assigns a cycle number to this thread
      int bowl;   //Stores current bowl in use

      //Avoid unused variable warnings.
      (void) unusedpointer;

      //Mice begin or stop
      P(mousesemaphore);

      //Mouse starts eating
      P(bowlsem); //Take one bowl

      miceeating++;

      if(miceeating == 1)
      {
            P(printsem);
            kprintf("Mice are starting to eat\n");
            V(printsem);
      }

      for(i = 0; i < NFOODBOWLS; i++) //Used to assign bowl #
      {
            if(bowllist[i] == 1)
            {
                  bowl = i; //This mouse takes a bowl
                  bowllist[i] = 0; //Remove bowl from list
                  break;
            }
      }
      P(printsem);
      c = cycle++;
      kprintf("Cycle %u: Mouse %u beginning to eat at bowl %u\n", c,
mousenumber, bowl+1);
      V(printsem);
      //Simulate Eating Time, no cats can be eating
      for(i = 0; i < 5000; i++)
            assert(catseating == 0);
      //Place bowl back into list
      bowllist[bowl] = 1;
      P(printsem);
      kprintf("Cycle %u: Mouse %u finishing eating at bowl %u\n", c,
mousenumber, bowl+1);
      V(printsem);
      //Mouse Finished eating
      micedoneeating++;
      miceeating--;
      V(bowlsem); //Replace one bowl

      //True when last mousesem thread executes this line of code
      if(micedoneeating == NMICE)
      {
            //Mousesem complete, all mice have finished eating
            P(printsem);
            kprintf("All %u mice have finished eating\n",
micedoneeating);
            V(printsem);
```

```
            //Allow cats to eat now if they havn't
            if(catsdoneeating == 0)
            {
                  for(i = 0; i < NCATS; i++)
                        V(catsemaphore);
            }
      }
}


/*
 * catmousesem()
 *
 * Arguments:
 *      int nargs: unused.
 *      char ** args: unused.
 *
 * Returns:
 *      0 on success.
 *
 * Notes:
 *      Mice or cats may start first, but you must initialize one or
 *    the other appropriatly in the initialization function.
 *    -To initialize cats first catsemaphore must be set to NCATS
 *    and mousesemaphore to 0.
 *    -To initialize mice first mousesemaphore must be set to NMICE
 *    and catsemaphore to 0.
 */

int
catmousesem(int nargs,
            char ** args)
{
      int index, error;

      // Initialize static semaphores to have mice eat first
      // Code also works if you make cats eat first.
      bowlsem = sem_create("bowlsem", NFOODBOWLS);
      catsemaphore = sem_create("catsemaphore", 0);
      mousesemaphore = sem_create("mousesemaphore", NMICE);
      printsem = sem_create("printsem", 1);

      // Initialize static variables
      cycle = 1;
      catseating = 0;
      miceeating = 0;
      catsdoneeating = 0;
      micedoneeating = 0;

      // Initialize static foodbowl array
      for (index = 0; index < NFOODBOWLS; index++)
            bowllist[index] = 1; //Sets all foodbowls as available
      /*
       * Avoid unused variable warnings.
       */

      (void) nargs;
      (void) args;

      /*
       * Start NCATS catsem() threads.
```

```
	 */

	for (index = 0; index < NCATS; index++)
	{
		error = thread_fork("catsem Thread",
				    NULL, index, catsem, NULL);

		/*
		 * panic() on error.
		 */

		if (error)
			panic("catsem: thread_fork failed: %s\n",
			      strerror(error));
	}

	/*
	 * Start NMICE mousesem() threads.
	 */

	for (index = 0; index < NMICE; index++)
	{
		error = thread_fork("mousesem Thread",
				    NULL, index, mousesem, NULL);

		/*
		 * panic() on error.
		 */

		if (error)
			panic("mousesem: thread_fork failed: %s\n",
			      strerror(error));
	}
	return 0;
}


/*
 * End of catsem.c
 */
```