Assignment 3 Solutions

cpe 453 Winter 2023

```
The men sat sipping their tea in silence.  After a while the klutz said,
        "Life is like a bowl of sour cream."
        "Like a bowl of sour cream?" asked the other.  "Why?"
        "How should I know?  What am I, a philosopher?"
```

— /usr/games/fortune

Not due at all—this one's optional—because of the weird start to the quarter, but it's not a bad exercise for exploring concurrent programming and it's not hard.

Program: dine

Using the pthreads library, implement a version of the Dining Philosophers Problem (described on p.66 in Tannenbaum and Woodhull and below).

The Dining Philosophers is a classic interprocess communication problem posed by Dijkstra in 1965: Five philosophers sit around a table. Between each pair of philosophers is a fork, and in the center of the table is a bowl of spaghetti. The life of a philosopher is very simple. It consists of alternate periods of eating and thinking:

To eat a philosopher requires two forks (either the spaghetti is very slippery, or the philosophers are clumsy). The forks can only be picked up one at a time, so he or she must grab one first, then the other. If a neighbor is holding one of the forks, the philosopher must wait until it has been set down. Once a philosopher has a fork, he or she will not relinquish it until after eating.

Upon gaining possession of both forks, a philosopher eats until full, then gets ready to think.

To think a philosopher puts down both forks, one at a time, then drops off into contemplation until hungry at which point the process starts again.

For the purposes of this assignment, a philosopher can be considered as always being in one of three states: `eating`, `thinking`, or `changing`. The first two states are self-explanatory. The third, `changing`, describes a philosopher who has decided to eat, but who has not yet acquired both forks, or who has decided to think, but who has not yet set down both forks.

All philosophers start out in the `changing` state, and start out hungry. That is, they will all try to eat before beginning to think.

Program requirements:

- While Dijkstra always had five philosophers, your program must be parametereized in terms of a constant, `NUM_PHILOSOPHERS`, that determines the number of philosophers. (Default 5) If the number is greater than the number of letters in the alphabet, just continue on up through the ASCII table for labels[1].

  To make it easy to change this value at compile time, please define this constant as shown below:

[1] That is, use 'A' plus i for philosopher i. This will lead to some weird characters at big parties, but you often meet weird characters at big parties.

```
#ifndef NUM_PHILOSOPHERS
#define NUM_PHILOSOPHERS 5
#endif
```

- Your program must use POSIX semaphores (see sem_overview(7)) to control individual forks between the philosophers. That is, you may not use the approach shown in the text.
- Your program must avoid deadlock, and, of course, must ensure that neighboring philosophers are never eating simultaneously.
- You should permit two non-adjacent philosophers to eat at the same time.
- To make the program runs more interesting, have the philosophers linger over their spaghetti for a random amount of time. See *Tricks and Tools* below.
- Each time a philosopher changes state, print a status line—in the format shown below—indicating what has happened. This status line should show the state of each philosopher ("Eat", "Think", or blank for changing), and the forks held by that philosopher.

  State changes include:

  – changing among "eat", "think" and "transition"
  – picking up a fork
  – setting down a fork
- You must take as an optional command-line argument an integer indicating how many times each philosopher should go through his or her eat-think cycle before exiting.

  Absent this argument, the value should default to 1.
- To make matters simple, number the forks 0,1,2,3,4, and the philosophers, named A,B,C,D, and E, sit at positions 0.5, 1.5, 2.5, 3.5, and 4.5.

  That is, philosopher A uses forks 0 and 1, philosopher B uses forks 1 and 2, etc.
- you will probably need a binary semaphore around the state printing and updating to ensure that it always prints a consistent state.
- **Write robust code.** Your code should compile cleanly with `-Wall`, and should not crash under **any** error conditions. If it cannot complete its task, it should print an error message and exit gracefully with non-zero exit status.
- **Be careful about synchronization; it's more complicated that it looks**.

Tricks and Tools

Using Pthreads

POSIX Threads (pthreads) is a threading library that can be used on many different platforms. It provides functions for thread creation and termination analogous to `fork(2)`, `exit(3)`, and `wait(2)`.

Be careful when using these functions: most of them pass references to other things. Consider the prototype of `pthread_create()`:

```
int pthread_create(  pthread_t * thread,
                     pthread_attr_t * attr,
                     void * (*start_routine)(void *),
                     void * arg);
```

The first argument is a pointer to a thread descriptor that will be set up by the call to describe the new thread. The second is a pointer to a set of attributes if you want to do something special. For this assignment, `NULL` is sufficient to take the defaults. The third is a function pointer to a function that takes a void pointer and returns a void pointer. The fourth pointer is the parameter to be passed to the function when the thread is started.

Note: void pointers are ANSI C's mechanism for dealing with generic pointers. C does not permit one to dereference a void pointer, so it is necessary to cast it to another pointer type first. Given a void pointer called `ptr`, it can be dereferenced into an int via: `int i = *(int *)ptr;`

The necessary pthreads functions are described in Table 1. Look at the man pages for full descriptions. When using pthreads, be sure to link with the pthreads library by adding "`-lpthread`" to your link line.

Because an example is worth a thousand words (well, 504 in this case), I have included a pthreads implementation of `trivial` in Figure 1.

`sem_overview(7)`	Provides an overview of POSIX semaphores.
`sem_wait(3)`	Decrement a semaphore or block trying. The DOWN() operation.
`sem_post(3)`	Wake a thread blocked on this semaphore if there are any, or increment the semaphore if not. The UP() operation.
`sem_init(3)`	Initialize a pthread semaphore for use. Takes a pointer to the semaphore, a flag determining how the semaphore is to be shared, and an initial value.
`sem_destroy(3)`	destroy (deallocate) a pthread semaphore. Takes a pointer to the semaphore.
`pthread_create(3);`	creates a pthread.
`pthread_join(3);`	the pthread equivalent of wait(2).
`pthread_exit(3);`	the pthread equivalent of exit(3).

Table 1: Useful pthreads functions

Some Notes

There are a few peculiarities about pthreads you ought to know:

- Be sure to include `-lpthread` on your link line.

Avoiding Deadlock

The dining philosophers problem is interesting because of the ways in which it can fail. If each philosopher simultaneously picks up the fork to his or her left, there will be no right-hand forks available and no philosopher will be able to get two forks. Since no philosopher will set down a fork in hand, they will all starve to death at the table. In terms of operating systems, we would say that the philosopher processes have deadlocked.

In order to avoid deadlock it is necessary to ensure that such a circular wait cannot happen. It turns out that in the case of the dining philosophers all that is necessary is to break the symmetry.

```c
/*
 * trivial_pt.c is a re-make of asgn1's fork-testing program
 * done as as demonstration of pthreads.
 *
 * -PLN
 *
 * Requres the pthread library.  To compile:
 *   gcc -Wall -lpthread -o trivial_pt trivial_pt.c
 *
 * Revision History:
 *
 * $Log: trivial_pt.c,v $
 * Revision 1.1  2003-01-28 12:55:00-08  pnico
 * Initial revision
 *
 *
 *
 */
#include <errno.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>

#define NUM_CHILDREN 4

void *child(void *id) {
  /*
   * this function will be executed as the body of each child
   * thread.  It expects a single parameter that is a pointer
   * to an integer, its ID.
   *   The parameter is a void* to comply with the prototype,
   * but we know what's really in there.
   */
  int whoami=*(int*)id;          /* numeric id */

  printf("Child %d (%d):    Hello.\n\n",   whoami, (int) getpid());
  printf("Child %d (%d):    Goodbye.\n\n", whoami, (int) getpid());

  return NULL;                   /* exits the thread with no final message */
}


int main(int argc, char *argv[]) {
  pid_t ppid;
  int i;

  int       id[NUM_CHILDREN];       /* individual identifiers (see below) */
  pthread_t childid[NUM_CHILDREN]; /* ctivations for each child */

  /* initialize the parent process id for later use */
  ppid = getpid();

  /* initialize an array of ID numbers for the children.
   *   It would be tempting to just pass the loop index
   * (like we did with trivial), but a pointer is passed
   * to the new thread, not the argument itself.  Because
   * the loop index will change, the effect is not what
   * one would hope.
   *   So, we give each child its own _independent_ ID
   * in the id array.
   */
  for(i=0;i<NUM_CHILDREN;i++)
    id[i]=i;

  /* Spawn all the children */
  for (i=0;i<NUM_CHILDREN;i++) {
    /* pthread_create() launches a new thread running the function
     * child(), passes a poitner to the argument in id[i], and
     * places a thread identifier in childid[i].
     *
     * A note on C:  below, I write "&childid[i]" to indicate the
     * address of the i-th element of the array child, but I could
     * just as well used pointer arithmetic and written "childid+i".
     */
    int res;
    res = pthread_create(
                         &childid[i], /* where to put the identifier      */
                         NULL,        /* don't set any special properties */
                         child,       /* call the function child()        */
                         (void*) (&id[i])   /* pass the address of id[i]  */
                         );

    if ( -1 == res ) {          /*  there was an error */
      /* report the error condition */
      fprintf(stderr,"Child %i:  %s\n",i,strerror(res));
      exit(-1);                 /* bail out. */
    }
  }


  /* Say hello */
  printf("Parent  (%d):    Hello.\n\n",(int) ppid);

  /*
   * Now wait for each child thread to finish.
   *   Note: Unlike the original trivial, pthread_join()
   * requires us to name a specific thread to wait for, thus
   * the children will always join in the same order regardless
   * of when they actually terminate.
   */

  for(i = 0 ; i < NUM_CHILDREN ; i++ ) {
    pthread_join(childid[i],NULL);
    printf("Parent  (%d):    child %d exited.\n\n",
           (int) ppid, i);
  }

  /* Say goodbye */
  printf("Parent  (%d):    Goodbye.\n\n",(int) ppid);

  return 0;                     /* exit successfully */
}
```

Figure 1: A pthreads example: `trivial_pt`

If even philosophers try to pick up their right-hand forks first, and odd philosophers (aren't they all?) try to pick up their left-hand forks first, the problem is avoided.

You can use this information in your solution once you've convinced yourself that it's true.

Randomization

To make the behavior of the philosophers a little more interesting, have them linger a little while eating or thinking. The routine `dawdle()` shown in Figure 2, sleeps for a randomly chosen number of milliseconds up to a second. `dawdle()` must be linked with the realtime library by adding "`-lrt`" to the link line.

Note that `random(3)` will always produce the same pseudorandom sequence unless seeded with `srandom(3)` at the beginning of the run. My published solution uses the number of seconds since the epoch plus the number of microseconds since the last second as the seed. These values are available via `gettimeofday(2)`.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <limits.h>

void dawdle() {
  /*
   * sleep for a random amount of time between 0 and 999
   * milliseconds.  This routine is somewhat unreliable, since it
   * doesn't take into account the possiblity that the nanosleep
   * could be interrupted for some legitimate reason.
   *
   * nanosleep() is part of the realtime library and must be linked
   * with -lrt
   */
  struct timespec tv;
  int msec = (int)(((double)random() / RAND_MAX) * 1000);

  tv.tv_sec  = 0;
  tv.tv_nsec = 1000000 * msec;
  if ( -1 == nanosleep(&tv,NULL) ) {
    perror("nanosleep");
  }
}
```

Figure 2: A routine to sleep for a little while

Displaying the status

Every time something changes, it is important to report it so that the person running the program can see what has happened. The format should be as shown below, five columns showing the instantaneous status of all the philosophers. The status should show the philosopher's state (eating, thinking, or changing) and which forks the philosopher is holding.

How you implement this is, of course, up to you, but do remember that more than one philosopher may change state at a time, so you may have to place locks around your status printing routines in addition to the locks on the forks.

Coding Standards and Make

See the pages on coding standards and make on the cpe 453 class web page.

What to turn in

Submit via `handin` to the `asgn3` directory of the `pn-cs453` account:

- your well-documented source files.
- A makefile (called `Makefile`) that will build your program when given the target "`dine`".
- A README file that contains:
  - Your name.
  - Any special instructions for running your program.
  - Any other thing you want me to know while I am grading it.

Sample runs

Below are some sample runs of `dine`. I will also place executable versions on the CSL machines in `~pn-cs453/demos` so you can run it yourself.

```
% dine
|=============|=============|=============|=============|=============|
|      A      |      B      |      C      |      D      |      E      |
|=============|=============|=============|=============|=============|
| -----       | -----       | -----       | -----       | -----       |
| -1---       | -----       | -----       | -----       | -----       |
| 01---       | -----       | -----       | -----       | -----       |
| 01--- Eat   | -----       | -----       | -----       | -----       |
| 01--- Eat   | -----       | ---3-       | -----       | -----       |
| 01--- Eat   | -----       | --23-       | -----       | -----       |
| 01--- Eat   | -----       | --23- Eat   | -----       | -----       |
| 01---       | -----       | --23- Eat   | -----       | -----       |
| 0----       | -----       | --23- Eat   | -----       | -----       |
| -----       | -----       | --23- Eat   | -----       | -----       |
| -----       | -1---       | --23- Eat   | -----       | -----       |
| -----       | -1---       | --23- Eat   | -----       | 0----       |
| -----       | -1---       | --23- Eat   | -----       | 0---4       |
| -----       | -1---       | --23- Eat   | -----       | 0---4 Eat   |
| ----- Think | -1---       | --23- Eat   | -----       | 0---4 Eat   |
| ----- Think | -1---       | --23-       | -----       | 0---4 Eat   |
| ----- Think | -1---       | --2--       | -----       | 0---4 Eat   |
| ----- Think | -1---       | -----       | -----       | 0---4 Eat   |
| ----- Think | -1---       | -----       | ---3-       | 0---4 Eat   |
| ----- Think | -12--       | -----       | ---3-       | 0---4 Eat   |
| ----- Think | -12-- Eat   | -----       | ---3-       | 0---4 Eat   |
| ----- Think | -12-- Eat   | ----- Think | ---3-       | 0---4 Eat   |
| ----- Think | -12-- Eat   | ----- Think | ---3-       | 0---4       |
| ----- Think | -12-- Eat   | ----- Think | ---3-       | ----4       |
| ----- Think | -12-- Eat   | ----- Think | ---3-       | -----       |
| ----- Think | -12-- Eat   | ----- Think | ---3-       | ----- Think |
| ----- Think | -12-- Eat   | ----- Think | ---34       | ----- Think |
| ----- Think | -12-- Eat   | ----- Think | ---34 Eat   | ----- Think |
| ----- Think | -12--       | ----- Think | ---34 Eat   | ----- Think |
| ----- Think | --2--       | ----- Think | ---34 Eat   | ----- Think |
| ----- Think | -----       | ----- Think | ---34 Eat   | ----- Think |
| ----- Think | ----- Think | ----- Think | ---34 Eat   | ----- Think |
| ----- Think | ----- Think | ----- Think | ---34 Eat   | -----       |
| -----       | ----- Think | ----- Think | ---34 Eat   | -----       |
| -----       | ----- Think | -----       | ---34 Eat   | -----       |
| -----       | -----       | -----       | ---34 Eat   | -----       |
| -----       | -----       | -----       | ---34       | -----       |
| -----       | -----       | -----       | ----4       | -----       |
| -----       | -----       | -----       | -----       | -----       |
| -----       | -----       | -----       | ----- Think | -----       |
| -----       | -----       | -----       | -----       | -----       |
|=============|=============|=============|=============|=============|
```

```
% dine 2
|=============|=============|=============|=============|=============|
|      A      |      B      |      C      |      D      |      E      |
|=============|=============|=============|=============|=============|
| -----       | -----       | -----       | -----       | -----       |
| -1---       | -----       | -----       | -----       | -----       |
| 01---       | -----       | -----       | -----       | -----       |
| 01--- Eat   | -----       | -----       | -----       | -----       |
| 01--- Eat   | -----       | ---3-       | -----       | -----       |
| 01--- Eat   | -----       | --23-       | -----       | -----       |
| 01--- Eat   | -----       | --23- Eat   | -----       | -----       |
| 01---       | -----       | --23- Eat   | -----       | -----       |
| 0----       | -----       | --23- Eat   | -----       | -----       |
| -----       | -----       | --23- Eat   | -----       | -----       |
| ----- Think | -----       | --23- Eat   | -----       | -----       |
| ----- Think | -1---       | --23- Eat   | -----       | -----       |
| ----- Think | -1---       | --23- Eat   | -----       | 0----       |
| ----- Think | -1---       | --23- Eat   | -----       | 0---4       |
| ----- Think | -1---       | --23- Eat   | -----       | 0---4 Eat   |
| ----- Think | -1---       | --23-       | -----       | 0---4 Eat   |
| ----- Think | -1---       | --2--       | -----       | 0---4 Eat   |
| ----- Think | -1---       | -----       | -----       | 0---4 Eat   |
| ----- Think | -12--       | -----       | -----       | 0---4 Eat   |
| ----- Think | -12-- Eat   | -----       | -----       | 0---4 Eat   |
| ----- Think | -12-- Eat   | ----- Think | -----       | 0---4 Eat   |
| ----- Think | -12-- Eat   | ----- Think | ---3-       | 0---4 Eat   |
| -----       | -12-- Eat   | ----- Think | ---3-       | 0---4 Eat   |
| -----       | -12-- Eat   | -----       | ---3-       | 0---4 Eat   |
| -----       | -12--       | -----       | ---3-       | 0---4 Eat   |
| -----       | --2--       | -----       | ---3-       | 0---4 Eat   |
| -1---       | --2--       | -----       | ---3-       | 0---4 Eat   |
| -1---       | -----       | -----       | ---3-       | 0---4 Eat   |
| -1---       | ----- Think | -----       | ---3-       | 0---4 Eat   |
| -1---       | ----- Think | -----       | ---3-       | 0---4       |
| -1---       | ----- Think | -----       | ---3-       | ----4       |
| 01---       | ----- Think | -----       | ---3-       | ----4       |
| 01--- Eat   | ----- Think | -----       | ---3-       | ----4       |
| 01--- Eat   | ----- Think | -----       | ---3-       | -----       |
| 01--- Eat   | ----- Think | -----       | ---34       | -----       |
| 01--- Eat   | ----- Think | -----       | ---34 Eat   | -----       |
| 01--- Eat   | ----- Think | -----       | ---34 Eat   | ----- Think |
| 01--- Eat   | ----- Think | -----       | ---34 Eat   | -----       |
| 01--- Eat   | ----- Think | -----       | ---34       | -----       |
| 01--- Eat   | ----- Think | -----       | ----4       | -----       |
| 01--- Eat   | ----- Think | ---3-       | ----4       | -----       |
| 01--- Eat   | ----- Think | --23-       | ----4       | -----       |
| 01--- Eat   | ----- Think | --23- Eat   | ----4       | -----       |
| 01--- Eat   | ----- Think | --23- Eat   | -----       | -----       |
| 01--- Eat   | ----- Think | --23- Eat   | ----- Think | -----       |
| 01--- Eat   | -----       | --23- Eat   | ----- Think | -----       |
| 01---       | -----       | --23- Eat   | ----- Think | -----       |
| 0----       | -----       | --23- Eat   | ----- Think | -----       |
```

```
| -----       | -----       | --23- Eat   | ----- Think | -----       |
| ----- Think | -----       | --23- Eat   | ----- Think | -----       |
| ----- Think | -1---       | --23- Eat   | ----- Think | -----       |
| ----- Think | -1---       | --23- Eat   | ----- Think | 0----       |
| ----- Think | -1---       | --23- Eat   | ----- Think | 0---4       |
| ----- Think | -1---       | --23- Eat   | ----- Think | 0---4 Eat   |
| ----- Think | -1---       | --23-       | ----- Think | 0---4 Eat   |
| ----- Think | -1---       | --2--       | ----- Think | 0---4 Eat   |
| ----- Think | -1---       | -----       | ----- Think | 0---4 Eat   |
| ----- Think | -12--       | -----       | ----- Think | 0---4 Eat   |
| ----- Think | -12-- Eat   | -----       | ----- Think | 0---4 Eat   |
| ----- Think | -12-- Eat   | ----- Think | ----- Think | 0---4 Eat   |
| ----- Think | -12--       | ----- Think | ----- Think | 0---4 Eat   |
| ----- Think | --2--       | ----- Think | ----- Think | 0---4 Eat   |
| ----- Think | -----       | ----- Think | ----- Think | 0---4 Eat   |
| ----- Think | ----- Think | ----- Think | ----- Think | 0---4 Eat   |
| -----       | ----- Think | ----- Think | ----- Think | 0---4 Eat   |
| -----       | ----- Think | ----- Think | -----       | 0---4 Eat   |
| -----       | ----- Think | ----- Think | ---3-       | 0---4 Eat   |
| -----       | ----- Think | ----- Think | ---3-       | 0---4       |
| -----       | ----- Think | ----- Think | ---3-       | ----4       |
| -----       | ----- Think | ----- Think | ---3-       | -----       |
| -----       | ----- Think | ----- Think | ---34       | -----       |
| -----       | ----- Think | ----- Think | ---34 Eat   | -----       |
| -----       | ----- Think | ----- Think | ---34 Eat   | ----- Think |
| -----       | -----       | ----- Think | ---34 Eat   | ----- Think |
| -----       | -----       | -----       | ---34 Eat   | ----- Think |
| -----       | -----       | -----       | ---34 Eat   | -----       |
| -----       | -----       | -----       | ---34       | -----       |
| -----       | -----       | -----       | ----4       | -----       |
| -----       | -----       | -----       | -----       | -----       |
| -----       | -----       | -----       | ----- Think | -----       |
| -----       | -----       | -----       | -----       | -----       |
|=============|=============|=============|=============|=============|
%
```

Solution:

```
CC     = gcc

CFLAGS  = -Wall -g

LD     = gcc

LIBS    = -lpthread -lrt

LDFLAGS = $(LIBS) -g

PROG    = dine

OBJS    = dine.o util.o scoreboard.o

SRCS    = dine.c util.c scoreboard.c

HDRS    = util.h dine.h scoreboard.h

EXTRACLEAN = .dependlist

all:     $(PROG)

$(PROG): $(OBJS)
        $(LD) $(LDFLAGS) -o $(PROG) $(OBJS)

allclean: clean
        @rm -f $(EXTRACLEAN)

clean:
        rm -f $(OBJS) *~ TAGS

tar:   $(PROG).tar

$(PROG).tar: Makefile $(SRCS) $(HDRS)
        tar cvf $(PROG).tar Makefile $(SRCS) $(HDRS)

tags : $(SRCS) $(HDRS)
        etags $(SRCS) $(HDRS)

test:    $(PROG)
        $(PROG)

#
# The .dependlist structure keeps the dependencies generated by makedepend
# out of the makefile proper.   This makes it easier to remove them all
# at once and restart on a different platform.   (The includes may be
# different.)   The implicit dependency on .dependlist (for its inclusion)
# also means that the dependencies will be automatically regenerated if
# any of the sources or headers change
#
depend: .dependlist
        @echo Updating local dependencies.

.dependlist: $(SRCS) $(HDRS)
        @makedepend $(MKDEPFLAGS) -f - $(SRCS) > .dependlist

include .dependlist

update:
        scp -p Makefile $(SRCS) $(HDRS) falcon:tmp/foo
```

```
#ifndef DINE_H
#define DINE_H

#include <errno.h>

#define DOWN(s) if ( 0 != sem_wait(s) ) {\
    fprintf(stderr,"sem_wait:  %s (%s)\n",strerror(errno),__FUNCTION__); \
    exit(EXIT_FAILURE);                                                  \
  }

#define UP(s) if ( 0 != sem_post(s) ) {\
    fprintf(stderr,"sem_post:  %s (%s)\n",strerror(errno),__FUNCTION__); \
    exit(EXIT_FAILURE);                                                  \
  }


#endif
```

```
/*
 * dine.c is the core of a pthreads implementation of the
 * Dining Philosophers Problem to illustrate interprocess
 * communication.
 *
 * Revision History:
 *
 * $Log: dine.c,v $
 * Revision 1.5  2016-11-01 09:37:09-07  pnico
 * Summary: fixed the check on pthread_create()
 *
 * Revision 1.4  2016-10-28 10:43:19-07  pnico
 * Updating to semaphores
 *
 * Revision 1.3  2003-02-04 15:06:36-08  pnico
 * Prepared the relase version with "extra" explanation
 * for use as a class example.
 *
 *
 */
#include <errno.h>
#include <limits.h>
#include <stdio.h>
#include <string.h>
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>
#include <semaphore.h>
#include <sys/time.h>

#include "dine.h"
#include "util.h"
#include "scoreboard.h"

/*
 * encode the program's ID in the executable.  Usefull if you
 * ever need to identify the file using strings(1)
 */
const char *version="$Id:  dine.c,v 1.5 2016-11-01 09:37:09-07 pnico Exp $";

static void unimportant_initialization();

int needed_thoughts=1;          /* default to thinking once */

#define NUM_PHILOSOPHERS 5
#define LEFT(N)  (N)
#define RIGHT(N) ((N+1)%NUM_PHILOSOPHERS)

sem_t *dinnerfork;              /* initialized in main() */

void think(int who) {
  /* change status to thinking, hang out, then move on to the
   * next activity.
   */
  sb_change_status(who,THINKING);
  dawdle();
  sb_change_status(who,WAITING);
}

void eat(int who) {
  /* change status to eating, hang out, then move on to the
   * next activity.
   *   Assumes the calling philosopher is holding both forks.
   */
  sb_change_status(who,EATING);
  dawdle();
  sb_change_status(who,WAITING);
}

void take_fork(int which, int who) {

  /* pick up the given fork, then report having it to
```

```
    * the world.   This is safe, since no other philosopher
    * could report having the fork if it happens to get
    * interrupted between the two statements.
    */

  DOWN(dinnerfork+which);
  sb_takefork(who,which);
}

void release_fork(int which, int who) {
  /* Report the fork as dropped, then unlock its semaphore.
    * This is safe, because even if there is a context switch
    * in–between, another process could not pick up the fork
    * until the semaphore is unlocked.
    */
  sb_dropfork(who,which);
  UP(dinnerfork+which);
}

void *philosopher(void *id) {
  /*
    * philosopher() is the function that implements the life of
    * a philisopher.   It takes its ID as loops the given number of times
    * through the eat–think cycle.
    */
  int who = *(int*)id;
  int num;
  int first,second;

  /* We will break the symmetry below (and prevent deadlock) by
    * forcing odd–numbered philosophers to taker the left fork
    * first, and even numbered ones to take the right one first
    */
  if ( who % 2 == 1 ) {         /* odd numbered philosopher */
    first  = LEFT(who);
    second = RIGHT(who);
  } else {                      /* even–numbered */
    first  = RIGHT(who);
    second = LEFT(who);
  }

  /* Cycle through the eat–think cycle the requisite number
    * of times.
    */
  for (num = 0; num < needed_thoughts; num++) {
    /* take the forks.         */
    take_fork(first,who);
    take_fork(second,who);

    /* eat for a while ("Grub first, then ethics" ––– B. Brecht) */
    eat(who);

    /* release the forks */
    release_fork(first,who);
    release_fork(second,who);

    /* now think a bit. */
    think(who);
  }

  return NULL;
}


int main(int argc, char *argv[]){
  /* create NUM_PHILOSOPHERS mutexes (one for each fork) and
    * NUM_PHILOSOPHERS threads (one for each philosopher)
    * start them going.
    *
    * Recall that with pointer arithmetic, p+i is equivalent to
    * &(p[i])
    */
```

```
  int i;
  pthread_t *thinker;
  int *id;
  char *end;

  if ( argc == 2 ) {
    needed_thoughts = strtol(argv[1],&end,0);
    if ( *end || needed_thoughts < 0 ) {
      fprintf(stderr,"%s:  must be a nonnegative integer\n",argv[1]);
      exit(EXIT_FAILURE);
    }
  } else if (argc > 2) {
    fprintf(stderr,"usage:  %s [times around]\n",argv[0]);
    exit(EXIT_FAILURE);
  }

  unimportant_initialization(); /* set some things up */

  sb_init(NUM_PHILOSOPHERS);

  thinker    = (pthread_t*)safe_malloc(NUM_PHILOSOPHERS * sizeof(pthread_t));
  dinnerfork = (sem_t*)safe_malloc(NUM_PHILOSOPHERS * sizeof(sem_t));
  id         = (int*)safe_malloc(NUM_PHILOSOPHERS * sizeof(int));

  for(i=0;i<NUM_PHILOSOPHERS;i++){
    if ( 0 != sem_init(dinnerfork+i,0,1) ) {
      perror("sem_init");
      exit(EXIT_FAILURE);
    }
  }


  for(i = 0; i < NUM_PHILOSOPHERS; i++) {
    id[i]=i;
    if ( 0 != pthread_create(thinker+i,
                             NULL,
                             philosopher,
                             (void *)(id+i))) {
      fprintf(stderr,"thread %i:  %s\n",i,
              strerror(errno));
    }
  }

  for(i = 0; i < NUM_PHILOSOPHERS; i++ ) {
    pthread_join(thinker[i],NULL);
    /*      printf("Philosopher %d leaves the building.\n",i); */
  }

  /* deallocate all locks */
  for(i = 0; i < NUM_PHILOSOPHERS; i++ ) {
    if ( 0 != sem_destroy(dinnerfork+i) ) {
      perror("sem_destroy");
    }
  }

  /* shut down the scoreboard */
  sb_destroy();

  return 0;
}


static void unimportant_initialization() {
  /* It isn't really important, but this seeds the random
   * number generator with the number of seconds since 1/1/70
   * plus the number of microseconds since the last second
   *  This is not intended to be rigorous, but just to
   * perturb things a little.
   */
  struct timeval tv;
  if ( -1 == gettimeofday(&tv,NULL)) {
```

```
    perror("gettimeofday");     /* oh, well, no seeding for us. */
  } else {
    srandom(tv.tv_sec + tv.tv_usec);    /* seed random a bit... */
  }
}
```

```
#ifndef SCOREBOARD_H
#define SCOREBOARD_H


/* tags for each status */

#define WAITING  0
#define EATING   1
#define THINKING 2

/* utility definitions */
#ifndef TRUE
#define TRUE (1==1)
#endif

#ifndef FALSE
#define FALSE (0==1)
#endif

/* extern declarations for external use */
extern void sb_init(int size);
extern void sb_destroy();
extern void sb_change_status(int who, int status);
extern void sb_takefork(int who, int which);
extern void sb_dropfork(int who, int which);
#endif
```

```
/* a scoreboarding system for hungry philosophers
 *
 * A Philosopher's status will either be eating, thinking, or waiting,
 * represented by "E", "T", or " ".  A philosopher can be holding one
 * or two forks.
 *
 * Each time a status change occurs, a new status line is printed.
 *
 */

#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <semaphore.h>

#include "dine.h"
#include "scoreboard.h"
#include "util.h"

static void print_stat(int who);
static void sb_print_status();

static char *name[] = {"","Eat","Think"};

static sem_t statuslock;        /* to keep the scoreboard consistent */

struct sb_stat {
  int status;                   /* status of philosopher */
  int *fork;                    /* fork-vector */
};

static struct sb_stat *sb;
static int sb_size;

extern void sb_init(int size) {
  /* set up the scoreboard system */
  int i,j;

  sb_size = size;

  sb=(struct sb_stat*)safe_malloc(sb_size * sizeof(struct sb_stat));
  for(i=0;i<sb_size;i++) {
    sb[i].status = WAITING;
    sb[i].fork   = (int*)safe_malloc(sb_size * sizeof(int));
    for(j=0;j<sb_size;j++) {
      sb[i].fork[j]=FALSE;
    }
  }

  /* initialize the scoreboard semaphore */
  if (0 != sem_init(&statuslock,0,1)) {
    perror("sem_init");
    exit(EXIT_FAILURE);
  }

  /* after each is initialized, print the titles*/
  for(i=0;i<sb_size;i++)
    printf("|=============");
  printf("|\n");
  for(i=0;i<sb_size;i++)
    printf("|        %c        ",i+'A');
  printf("|\n");
  for(i=0;i<sb_size;i++)
    printf("|=============");
  printf("|\n");

  sb_print_status();

}

static void print_stat(int who) {
  int i;
```

```
  printf("| ");
  for(i=0;i<sb_size;i++) {
    if ( sb[who].fork[i] )
      printf("%d",i);
    else
      printf("-");
  }
  printf(" %-5s ",name[sb[who].status]);
}

extern void sb_destroy() {
  /* free everything and close out the scoreboard */
  int i;

  for(i=0;i<sb_size;i++){
    free(sb[i].fork);
    printf("|=============");
  }
  free(sb);
  if ( 0 != sem_destroy(&statuslock) ) {
    perror("sem_destroy");       /* oh, well.   We tried */
  }

  printf("|\n");
}

static void sb_print_status(){
  /* should only be called by someone holding the statuslock */
  int i;

  for(i=0;i<sb_size;i++)
    print_stat(i);
  printf("|\n");
}

extern void sb_change_status(int who, int status) {
  DOWN(&statuslock);
  sb[who].status = status;
  sb_print_status();
  UP(&statuslock);
}

extern void sb_takefork(int who, int which) {
  DOWN(&statuslock);
  sb[who].fork[which] = TRUE;
  sb_print_status();
  UP(&statuslock);
}

extern void sb_dropfork(int who, int which) {
  DOWN(&statuslock);
  sb[who].fork[which] = FALSE;
  sb_print_status();
  UP(&statuslock);
}
```

```
#ifndef UTIL_H
#define UTIL_H

extern void*safe_malloc(int size);
extern void dawdle();

#endif
```

```c
#include <errno.h>
#include <limits.h>
#include <stdio.h>
#include <string.h>
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>

void dawdle() {
  /*
   * sleep for a random amount of time between 0 and 999
   * milliseconds.  This routine is somewhat unreliable, since it
   * doesn't take into account the possiblity that the nanosleep
   * could be interrupted for some legitimate reason.
   */
  struct timespec tv;
  int msec = (int)((((double)random()) / RAND_MAX) * 1000);

  tv.tv_sec  = 0;
  tv.tv_nsec = 1000000 * msec;
  if ( -1 == nanosleep(&tv,NULL) ) {
    perror("nanosleep");
  }
}

void *safe_malloc ( int size ) {
  void *new;
  new = malloc(size);
  if ( new == NULL ) {
    perror(__FUNCTION__);
    exit(-1);
  }
  return new;
}
```