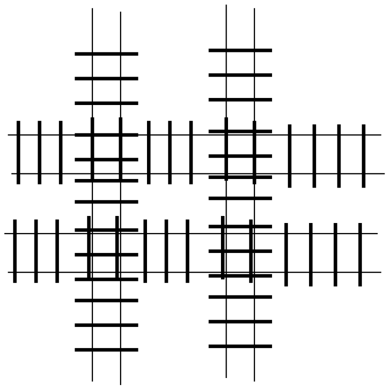
CS 511: Assignment #3
Traffic Management

Due Sunday, October 19 at 11pm.

1 Assignment

Consider the operation of a shipping warehouse. Workers in the storage areas put goods on motorized carts and program each cart's destination loading dock. The carts then travel along tracks to the loading docks, where their loads are placed onto trucks. The carts then return empty to the storage areas of the warehouse, where they receive a new load. The warehouse operates like this continually.

As shown in the figure below, there is a location in the warehouse where cart tracks from the four directions – north, south, east, west – meet at an intersection.



To provide for worker safety at most one cart may be in the intersection at a time (this way workers walking across the intersection need look out for only one direction at a time). As a cart approaches the intersection, control software decides whether to let it enter or to stop it. Any cart that is stopped before the intersection must eventually be allowed into the intersection; i.e., the system must be "live." Once in the intersection, a cart takes 10 seconds to pass through.

The rule that ensures liveness is that, after a cart passes through the intersection, the waiting cart nearest on the right is allowed to proceed next. For example, suppose that a cart coming from the north is passing through the intersection and that carts from the south and east are waiting to enter the intersection. After the cart from the north clears the intersection, the control software must allow one of the waiting carts to proceed. The first direction to the right of north is west; however, there are no carts from the west waiting. Proceeding rightward the next direction is south; carts from the south are waiting so the first of those is allowed to proceed into the intersection.

The assignment is to write the control software as a C program that uses locks, semaphores, and monitors as needed to ensure proper passage of carts through the intersection.

1.1 Software Details

You are given the definition of a cart in file `cart.h`. You are also given the "q" module in files `q.h` and `q.c`. The `q` module implements a simple ordered queue of carts. Function `q_putCart` places a cart at the tail of the queue. Function `q_getCart` takes a cart off the head of the queue. Function `q_print` prints the contents of the queue in order. You should create four queues to represent the carts coming from each direction.

You must write three files: `trafficmgr.c`, `monitor.h`, and `monitor.c`. You must also write a `Makefile` that will properly compile all the source into an executable named `trafficmgr`.

`trafficmgr` takes one command line argument, an arbitrary-length string composed from the letters n, s, e, and w. Each letter represents a cart from that direction that seeks to enter the intersection. For example, if `trafficmgr` is run as:

```
$ ./trafficmgr nsnewsn
```

then the first thing the program will do is populate four queues with the 7 carts and create four threads, one for each direction. Carts 1, 3, and 7 (in that order) will be coming from the north; carts 2 and 6 (in that order) will be coming from the south; cart 4 will be coming from the east; and cart 5 will be coming from the west. One of the first-arriving carts from each direction – 1, 2, 4, or 5 – will be first to enter the intersection; thereafter, the "next cart on the right" rule should prevail. For example, suppose that cart 4 from the east is the first into the intersection. After it passes through, the sequence of cart passages should be:

- Cart 1 from the north, because north is the nearest rightward direction from east and there are north carts waiting.
- Cart 5 from the west, because west is the nearest rightward direction from north and there is a west cart waiting.
- Cart 2 from the south, because south is the nearest rightward direction from west and there are south carts waiting.
- Cart 3 from the north, because there are no east carts waiting.
- Cart 6 from the south.
- Cart 7 from the north.

After all carts have passed through the intersection, the threads should be terminated, any resources (such as monitors, queues, semaphores, etc.) freed, and the program should terminate.

The monitor module that you write should have 5 functions:

- `monitor_init` – initialize monitor
- `monitor_arrive` – the actions of a cart before it enters the intersection
- `monitor_cross` – the actions of a cart as it crosses the intersection
- `monitor_leave` – the actions of a cart afters it passes through the intersection
- `monitor_shutdown` – free monitor resources

Accordingly, the action of each of the 4 threads should be like this:

```
fprintf(stderr, "thread for direction %c starts\n", direction);
cart = q_getCart(direction);
while (cart != NULL) {
    fprintf(stderr, "thread for direction %c gets cart %i\n",
                                           direction, cart->num);
    monitor_arrive(cart);
    monitor_cross(cart);
    monitor_leave(cart);
    cart = q_getCart(direction);
}
fprintf(stderr, "thread for direction %c exits\n", direction);
```

For debugging and grading purposes, your program should print informative messages when any of these events occurs:

- Thread starts
- Thread gets cart from head of queue
- Cart arrives at intersection
- Cart must wait before entering intersection
- Cart allowed to proceed into intersection
- Cart enters intersection
- Cart crosses intersection
- Cart leaves intersection
- Thread signals another (identify the direction associated with each thread)
- Thread ends

1.2 Programming Notes

Remember that when thread A signals thread B, there is no assurance that B will be the next thread to run. Depending on how you structure your program, this fact can frustrate your intended behavior.

There is no requirement to use barriers; nevertheless, you might want to learn about them by using a barrier to cause all threads to shut down at the same time. Barriers are simple to use and sometimes very useful. Read the man pages for `pthread_barrier_init`, `pthread_barrier_wait`, etc.

2 Submission Instructions

You may work alone or with a partner; the maximum group size is two.

Submit `trafficmgr.c`, `monitor.c`, `monitor.h` and `Makefile` via Moodle by the indicated date and time. Moodle will shut you out at the deadline. Since your clock and Moodle's clock may differ by a few minutes, be sure to submit at least several minutes before the deadline.

Please note: *late work is not accepted; late submissions will not be graded and will receive a score of zero.* Also, points will be deducted if files have names different from required.

Typing "`make`" on `linux-lab.cs.stevens.edu` should build `trafficmgr` with no errors or warnings using the maximum compiler settings of "`-Wall -pedantic-errors`"