CS 4/53201 Operating Systems

Project #2

Due via e-mail by 11:59pm on Monday, March 31, 2003

Preliminaries

Ohio Department of Transportation was not satisfied with the results of the recent construction project in the Main Street next to Kent State University and started the construction there again. Only one lane is left operational during the construction work. Therefore, the cars going in the opposite directions have to coordinate their movement. Besides, the students from the university like to go to the Susanne's coffee shop across the street for lunch. All this created quite an inconvenience for everybody, and ODOT installed traffic lights on both sides of the one-lane section of the road and by the pedestrian crosswalk. ODOT also hired you to write a program to coordinate the work of the traffic lights according to the following rules:

- cars cannot go in the opposite directions simultaneously on the one-lane section;
- a pedestrian cannot cross the street while there is a car in the one-lane section;
- a car may enter the one lane section if there is a car there going in the same direction;
- a car does not wait for more than two cars going in the opposite direction;
- a pedestrian has to yield to cars BUT a pedestrian should not wait for more than tree cars.

Programming Assignment

All changes you make to Nachos source code must be clearly marked as follows:

```
//project 2 changes start here
<put your changes here>
//project 2 changes end here
```

Implement locks and condition variables that are defined (but not implemented) in `threads/synch.h` and `threads/synch.cc`. Use your implementation in the assignment below.

Modify Nachos `main()` function so that it accepts a new option "`-a`" (you can reuse your solution for Project #1). This option can be followed by a phrase of arbitrary length. For example, the nachos executable can be run as:

```
prompt% nachos –a E1 W1 W2 P1 E2 E3 E4 W3 W4 P2
```

The first letter of every word is ether E – car going East, W – car going West, or P – pedestrian. Write a function `Street()` that forks a thread for every car or pedestrian. A car thread executes a function `CarWating()` indicating that it wants to enter the one lane section of the street; a pedestrian thread executes a function `PedWating()` to cross the street. You have to write the functions so that the thread is blocked if it is not safe to enter the street according to the rules above. When the function exits the car or pedestrian thread prints:

<name of the thread> `entering street`

where <name of the thread> is the word from the command line that identified the thread. Then the thread executes a `Yield()` to simulate possible concurrent execution of multiple threads. Then the thread prints:

<name of the thread> `exiting street`

and executes the function `CarOut()` or `PedOut()` indicating that the thread exited the street. After returning from this function a thread terminates. The `CarOut()` and `PedOut()` should unblock the threads waiting to enter the street according to the rules listed above. Note that it has to unblock multiple threads if it is safe for them to enter the street (several pedestrians, cars going in the same direction).

Note that you may not use one critical section for entering and exiting from the street because there may be more threads in the street simultaneously.

The output of your program should look similar to the following.

```
prompt% nachos -a E1 W1 P1 W2 E2 E3 E4 W3 W4
Entering main*** thread 0 looped 0 times
*** thread 1 looped 0 times
[…]
*** thread 1 looped 4 times
E1 entering street
E1 exiting street
W1 entering street
W1 exiting street
W2 entering street
W2 exiting street
[…]
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!
[…]
Cleaning up...
prompt%
```

Submitting Your Project

You have to submit your project to the teaching assistant (Meiduo Wu `mwu@mcs.kent.edu`) by e-mail. She will acknowledge the receipt of your project within a day. Send **only** the files you modified – for example `thread.cc` and `thread.h`. Your submission should be in the form of a tar archive. The command to create the archive file (named `project2.tar` containing files `thread.cc`, `thread.h` and `answers.txt`) is as follows:

```
tar cvf project2.tar thread.cc thread.h answers.txt
```

Warning: after you have submitted your project, do NOT touch or modify the files you submitted. If there is a problem with the submission, you will be asked to produce the files and the modification time will be used to determine the time of the submission.
Partial submissions are not accepted.

Grading

The programming project is worth 50 points. Late projects are accepted. Late policies are outlined in the syllabus.