Programming Assignment IV

CS3331

Spring 2025

1 Party Room

A landlord in a college town has a large apartment with a fantastic social room. Many of the students at the local university stay in this apartment, and are prone to have wild parties in the aforementioned social room. These massive parties frequently result in noise complaints from the building's neighbors. As such, the landlord regularly visits the building to check for and break up any ongoing parties in the social room.

The landlord is modeled as a thread with the following pattern. After checking the room m times, the landlord has decided he has had enough, and that the backbreaking thankless job of renting out apartments is more than he can handle. Thus, after m iterations, he evicts all the tenants, sells his apartment, and goes to live a modest retirement in the Bahamas.

```
for (i = 0; i < m; i++) { // m is an input value
    ...
    Delay();              // rest for awhile
    CheckRoom(...);       // check the social room
    Delay();              // rest for awhile
    ...
}
```

CheckRoom() is a function to model the landlord checking the social room based on the above rules.

Each student is also modeled as a thread:

```
while (1) {
    ...
    Delay();          // study for a while
    GoToParty(...);   // go to the party
    Delay();          // go sleep the party off after leaving
    ...
}
```

Your job is to fill all the needed synchronization and other activities into two functions CheckRoom() and GoToParty(). Your implementation must satisfy specification given in section 1.3.

1.1 Input

The input to your program is given as three command line arguments, like so:

```
./prog4  m n k
```

where

- m is the number of times the landlord will check the room before retiring,
- n is the maximum number of students allowed in the room before the landlord considers it a party, and
- k is the total number of students in the apartment.

You may assume that $30 \geq m > 0$, and $30 \geq k \geq n > 0$

1.2 Output

Any ith student thread will produce a number of different print statements:

1. The student thread will print the following when the thread if first spawned

   `Student i starts`

2. The student will print the following when she wants to enter the social room

   `Student i waits to enter the room`

3. The student will print the following when she finally able to enter the room

   `Student i enters the room (x students in the room)`

   where x is the number of students in the room after she enters (inclusive of the student that just entered)

4. A student will print this when she is waiting to leave the party (either voluntarily, or because she is being kicked out by the landlord)

   `Student i waits to leave the room`

5. Finally, a student will print this when she leaves the room

   `Student i leaves the room (x students in the room)`

   here x is the number of students in the room after she leaves.

6. After the m cycles have been performed, students will print

   `Student i is evicted and terminates`

   The student can only terminate outside of the party room.

The landlord is a little more complex

1. The landlord, like students, prints a message when the thread is spawned

   `The landlord starts`

2. When the landlord enters the room to check it for the yth time, this message will be printed:

   `The landlord checks the room the yth time`

   After this is printed, the landlord will check the room condition and print one of the following:

   (a) If the landlord finds there are no students in the room, he will print:

   `The landlord finds the room has no students and leaves`

   (b) If the landlord finds there is an acceptable number ($x \leq n$) of students in the room, he prints:

   `The landlord finds there are only x students in the room and leaves`

   (c) If the landlord finds there are enough students in the room ($x > n$) for it to be a party, he prints

```
The landlord finds x students in the room and starts breaking up the party
```

3. If the landlord found there was a party in the room, he will go around breaking up the party. When he has finished, he will print:

   ```
   The landlord finishes breaking up the party and forces everyone to leave
   ```

4. After the landlord finishes clearing the room out, and all the students who were in the room have left, he will print:

   ```
   The landlord leaves the room after clearing it of party-goers
   ```

5. After checking the room for the mth time, the landlord decides to sell the apartment and evict all the tenants.

   ```
   After checking the room m times, the landlord decides to evict all the students
   ```

6. Before the landlord can evict students, he must first finish clearing the room

   ```
   The landlord finishes clearing the room for the last time, and now evicts the students
   ```

7. Once all tenants have been evicted, the landlord goes to retire, and terminates.

   ```
   Having evicted all the students, the landlord sells the building and retires
   ```

   This must be the last statement printed by the program.

The landlord's print statements begin on column 1 (i.e., 0 leading spaces), and the ith student's print statements begin on the $(i+1)$th column, (i.e., i leading spaces). Please see the example output included on the assignment's canvas page for a complete output example.

1.3 Specification

Below is a more formal specification of the behavior of the program. $S.i$ refers the to ith type of student print statement given in section 1.2; similarly, $L.i$ labels refer to the landlord's print statements.

1. Any number of students $\leq k$ can be in the room at the same time.
2. The room starts empty.
3. Students may enter ($S.3$) or exit ($S.5$) the room freely, unless the landlord is in the room.
4. The landlord can always enter the room ($L.2$), so long as he is not already in the room.
5. If the landlord sees there are more than n students are in the room, he will break up the party.
6. If the landlord breaks up the party, he will not leave ($L.4$) until all students have left the room.
7. After the landlords mth time checking the room, he will clear it, regardless of how many students are in the room ($L.5$).
8. After the landlord has cleared the room for the last time ($L.6$), students may not enter the room ($S.2$).
9. Students may only wait to enter the room ($S.2$) if they are outside it.
10. Students may only wait to leave the room ($S.4$) if they are inside it.
11. Students may only enter the room ($S.3$) after they have waited to enter the room ($S.2$), and leave ($S.5$) after they have waited to leave the room ($S.4$).
12. Students cannot enter the room ($S.3$) while the landlord is inside the room ($L.2$).
13. Students cannot leave the room ($S.5$) while the landlord is inside the room ($L.2$), unless the landlord has started clearing the room ($L.3 \vee L.5$).

14. Students may only be evicted ($S.6$) while they are outside the room.
15. Students may not print anything else after they have been evicted ($S.6$).
16. The landlord cannot retire ($L.7$) until all students have been evicted ($S.6$).
17. The landlord retiring ($L.7$) is the last statement printed by the program.

2 Submission Guidelines

The following sections elaborate submission requirements not directly related to the semantics of the program itself.

2.1 General Rules

1. The program must be written in C++, programs written in other languages will receive a 0.
2. The program must use ThreadMentor, programs that make use of non-ThreadMentor concurrency/synchronization libraries (such as pthreads or `std::thread`) will receive a 0.
3. Your work must be submitted as a `.zip` file via canvas. This `.zip` will contain your source code, your `Makefile`, and your `README`.
4. Your `Makefile`, when passed the `noVisual` target, should produce an executable file named `prog4` using the non-visual version of ThreadMentor.
5. We will use the following approach to compile and test your program:

   ```
   make noVisual
   ./prog4 m n k
   ```

   (m, n, and k are placeholder values) This procedure may be repeated a number of times with different input configurations to see if your program behaves correctly.
6. Your implementation should fulfill the program specification as stated.
7. You should insert calls to ThreadMentor's `Delay()` function to increase the nondeterministic behavior of the system (this is good for testing your program!). Make sure you use ThreadMentor's `Delay()` function (which is a simulational delay, rather than physical) instead of `sleep()` and its ilk, as those will significantly slow down the execution of your program, and may result in the program being timed out by the grading script!
8. Your implementation should avoid busy waiting as a strategy for synchronization. Use semaphores instead!

2.2 Program Style and Documentation

- At the top of each `.cpp`/`.h` file, the first comment should be a program header to identify yourself like so:

  ```
  // ------------------------------------------------------
  // NAME : John Smith                          User ID: xxxxxxxx
  // DUE DATE : mm/dd/yyyy
  // PROGRAM ASSIGNMENT #n
  // FILE NAME : xxxx.yyy
  // PROGRAM PURPOSE :
  // A couple of lines describing your program briefly
  // ------------------------------------------------------
  ```

Here, User ID is the one you use to log in to MTU systems.

- Your programs must contain a reasonable amount of concise and to-the-point comments. Do not write a novel.
- Your program should have reasonable indentation. Pyramids of whitespace are hard to parse!

2.3 The `README` File

A `README` file should be included with your submission. This file must either be a plaintext file named `README`, or a PDF file named `README.pdf`. The `README` should answer the questions enumerated below.

Each of these questions asks about how you ensure your program satisfies a certain requirement given in section 1.3

1. How do you ensure specification #12, "Students cannot enter the room ($S.3$) while the landlord is inside the room ($L.2$)"?
2. How do you ensure specification #6, "If the landlord breaks up the party, he will not leave ($L.4$) until all students have left the room"?
3. How do you ensure specification #17, "The landlord retiring ($L.7$) is the last statement printed by the program"?

Please ensure you clearly indicate which question is being answered at a given point within your `README`. While a formal proof is not necessary, please give a convincing argument for each question. Arguments like "This situation cannot occur because I use a semaphore!" are not convincing. *How* does the semaphore prevent the bad state from occurring?

2.4 Submission File Structure

Your submission should follow this file structure:

```
submission/
  thread.h              # thread definitions
  thread.cpp            # thread (student and landlord) implementations
  thread-main.cpp       # main program implementation
  README{.pdf,.txt,}
  Makefile
```

Please do not write code in files outside the ones listed above, as you may lose points for not following the program file structure. Remember that file names in UNIX-like systems are case-sensitive, so `README.txt` is a different file from `readme.txt`!

3 Collaboration Policy

"Empty Hands" collaboration is permitted for this programming assignment.

4 Final Remarks

As I'm sure you're aware by now, deadlocks and race conditions can be very difficult to diagnose! Thus, start early to give yourself time to diagnose these issues if you run into them while implementing this program. While archaic, ThreadMentor's visual system can help you diagnose deadlocks, so it may help to familiarize yourself with it.

Finally, run through this quick checklist before and after you submit!

- My submission consists entirely of my own work, and I know I risk failing the entire course if I have plagiarized from any source.
- I am submitting the correct program to the correct assignment on canvas.
- My program follows the program file structure described in 2.4.
- I have included a `Makefile`.
- I have included my `README` file, and it is either a plaintext or PDF file.
- I have tested to make sure my `Makefile` works correctly, and successfully compiles the program using the command given in section 2.1.
- The included `Makefile` compiles my executable without the ThreadMentor visual system when passed the `noVisual` target.
- I have tested my program to ensure it works with several inputs, including the given example input.
- My program's output conforms to the output requirements described in section 1.2.
- My program takes input as command line arguments, as described in section 1.1.
- My program is written in C++.
- My program uses only ThreadMentor for concurrency and synchronization. I do not use pthreads or any other threading library.
- I have tested both compiling and running my program on a CS lab machine, or `colossus.it.mtu.edu`.
- My program runs threads concurrently, I haven't accidentally sequentialized the computation!