Programming Assignment (PA) - 3
Demo Session Simulation

CS307 - Operating Systems (Spring 2023)

3 May 2023
DEADLINE: 18 May 2023, 23:55

1 Introduction

As Sabanci University (SU) students, you are already familiar with demo sessions for class projects of Computer Science (CS) courses. In PA3 you will implement a C or C++ program that simulates a demo room.

Your program will be multi-threaded and threads will represent *participants*. There are two types of participants: *Assistants* and *Students*. Each participant/thread will arrive, enter the classroom, participate in a demo and leave. These actions will be simulated by printing prompts to the console.

Demo room has some rules that requires synchronization of participant threads. There are two main rules:

1. Inside the demo room, the number of students must be lower or equal to three times the number of assistants (e.g. if there are two assistants inside the room, there can be at most 6 students with them).

2. Each demo session will be performed by exactly three participants: one assistant and two students. There can be multiple active demo session in the room at any time.

Depending on these rules, there are four synchronization points for participants where they might block:

1. A student might block before entering the demo room, if there are already three times more students than assistants.

2. A participant (student or assistant) might block in the demo room if there are not enough participants (two students, one assistant) in the room for forming a new demo group.

3. Participants in an active demo session must block until everyone in the demo group declares his/her participation.

4. An assistant might block before leaving the room if the number of students in the room will exceed three times the number of assistants in the room when she leaves.

We expect you to solve this synchronization problems using semaphores (primary choice) or condition variables (secondary choice). We do not want you to implement blocked participants busy-waiting in a loop. If you use only mutexes for synchronization, having busy-waiting loops is inevitable. If you do so, you will lose some points (see Section 6). Each thread will check the blocking conditions itself and block itself using a semaphore or condition variable operation.

For this PA, it will be possible to get partial grades by implementing a subset of restrictions. For instance, if your program only implements the first restriction, you can get up to 80 points. We divide the second rule into simpler rules and allow finer-grained partial grading. For details, please consider Section 5.

2 Threads

You are required to create a POSIX thread (pthread) for participants. You can make assistants and students run the same method, but we heavily suggest that you implement different methods for them. In addition to participant threads, there will be a main thread creating participant threads.

2.1 The Main Thread

The main thread, which is the thread of the process created by the operating system when the application starts, runs the main method.

The main method will require two arguments to start and function properly. Since there are assistants and students, there will be two integer arguments representing the number of them. These arguments will be program input when called from the console. An example call of your program is as follows:

```
$ ./demosim 2 4
```

where `demosim` is your executable (compiled version of your source code), 2 and 4 are the number of assistants and students, respectively.

In the main method, you have to check validity of arguments as well. There are two conditions on the number of participants:

- The number of assistants must be positive.
- The number of students must be two times the number of assistants.

These conditions guarantee that all of the assistants and students can eventually enter, participate in the demo session and leave.

If arguments are valid, then the main method creates the participant threads according to the given arguments. Then, it waits until all of the participant threads terminate. Otherwise, the main thread terminates before creating any child thread. In both cases, it must print `The main terminates.` at the end (after all children (if exists) terminate).

2.2 Student Threads

- It first prints: `Thread ID:` $< tid >$`, Role:Student, I want to enter the classroom.`
- If the number of students in the classroom are less then 3 times the number of assistants also in the classroom, it can enter. After entering, it will print `Thread ID:` $< tid >$`, Role:Student, I entered the classroom.`
- Inside the classroom, it checks if there is at least one other student and one assistant in the room which is not a part of a demo group, then it wakes them up. Otherwise it blocks/waits until a demo group including it is formed and someone in the group wakes it up.
- At this point, this thread is part of an active demo group and ready to participate in the demo session. It will print `Thread ID:` $< tid >$`, Role:Student, I am now participating.`
- After printing the previous line, it must block until other participants in the same demo group print their ready prompts. Then, it can leave the classroom by printing `Thread ID:` $< tid >$`, Role:Student, I am leaving.`.

2.3 Assistant Threads

- It first prints: `Thread ID:` $< tid >$`, Role:Assistant, I entered the classroom.`
- Inside the classroom it checks if there are at least two students already waiting, and if so it wakes two of them. Otherwise it blocks/waits until a demo group is formed and someone in the demo group wakes it up.
- At this point, this thread is part of an active demo group and ready to participate in the demo session. It will print `Thread ID:` $< tid >$`, Role:Assistant, I am now participating.`

- It blocks until all the students in its demo group announce their participation. Then, it will print `Thread ID:` $<tid>$`, Role:Assistant, demo is over.`

- Lastly, it tries to leave by checking whether leaving breaks the first condition: number of students must be lower than 3 times the number of assistants minus 3 Otherwise, it blocks until the condition is satisfied. When it can leave, it prints `Thread ID:` $<tid>$`, Role:Assistant, I left the classroom.` and terminates.

3 Implementation Details

As indicated above, you will use POSIX threads. In order to handle the synchronization issues you can use pthread semaphores and pthread barriers or you can implement your own barrier.

The pthread semaphores behave like Linux Zemaphores, not like the original Dijkstra semaphores we have seen in the lectures. So, before using the pthread semaphores, take a look at the manual page of the command `sem_wait` or check the web page : `https://linux.die.net/man/7/sem_overview`. If you wish, you can implement Dijkstra semaphores using pthread condition variables and mutexes as in the lectures. The choice is yours. In the report you will provide in the submission package, you have to explain which synchronization primitives you used and how you implemented them.

4 Correctness

There are correctness specifications for the main thread, student threads and assistant threads.

The correctness of the main thread is mostly explained in the Section 2.1. First, it should check validity of input arguments as described. If arguments are valid, then it should try creating the correct number of children threads. You do not have to consider the case when a child thread creation fails. However, you have to ensure that the main thread waits until all children terminate if all children are created successfully. The console prompt `The main terminates` by the main thread must always be printed as the last thing to the console.

Correctness of the student and assistant threads depends on the order and interleaving of strings they print to the console. To make things easier, let us give strings they print to the console at various steps and number of participants some names:

- num_s: The number of students
- num_a: The number of assistants
- $arrive$: The string `Thread ID:` $< tid >$`, Role:Student, I want to enter the classroom.`
- $enter$: The string `Thread ID:` $< tid >$`, Role:<Student/Assistant>, I entered the classroom.`
- $demo$: The string `Thread ID:` $< tid >$`, Role:<Student/Assistant>, I am now participating.`
- end: The string `Thread ID:` $< tid >$`, Role:Assistant, demo is over.`
- $leave$: The string `Thread ID:` $< tid >$`, Role:<Student/Assistant>, I left the classroom.`
  - For string str, str_s is its student version and str_a is its assistant version. For example, $leave_s$ is the string `Thread ID:` $< tid >$`, Role:Student, I left the classroom.`
  - For string str, $\#str$ is the number of occurrences of that string.

Then, for any execution of your program with valid inputs, the following conditions must be satisfied:

- For each student thread $arrive$, $enter$, $demo$ and $leave$ must be printed to the console in this order.
- Similarly for each assistant $enter$,$demo$,end,$leave$ must be printed to the console in this order.
- At any point, the number of students must be equal or lower to 3 times the number of assistants. To check this, the following formula must be valid on any prefix of a console output:
  $(\#enter_s - \#leave_s) \leq 3 * (\#enter_a - \#leave_a)$
- At any point, the number of $demo_s$s must be at least 2 times the number of ends. Similarly, the number of $demo_a$s must be at least equal to the number of ends.For this to be true, the following formula must be valid $\#2 * end \leq \#demo_s \wedge \#end \leq \#demo_a$

See Section 6, for a sample output obeying the correctness conditions above. **Note:** The grading for the last two items are separate.If you are aiming for partial points select one of them and be sure it is correct. If you implement them incompletely, you will most likely get 0 from both of them.

5 Partial Points

As you may have have guessed, this homework is rather challenging, so we provide some options to get partial points. The first one is skipping some of the rules. To recap, You have 3 rules:

1. Entering Condition (15 points)
2. Leaving Condition (15 points)
3. Grouping Condition (20 points)

According to the entering condition, a student can only enter if the number of students inside the classroom is smaller than or equal to the number of assistants inside the classroom times three. This corresponds to this formula: $\#enter_s \leq 3 * \#enter_a$ If you decide to not implement this, threads can enter the classroom whenever they want.

Similarly, leaving condition only permits an assistant to leave if the number of students that left the classroom is more than or equal to three times the number of students that left the classroom. This corresponds to this formula: $\#leave_s \geq 3 * \#leave_a$ If you decide to not implement this, treads can leave the classroom whenever they want.

Lastly, grouping condition checks whether a demo session is done by one assistant and two students. It also includes other demo instructions, like threads blocking until everyone declaring their participation. You can choose to not implement this, in which case you still need to print these lines according to in thread ordering, but student and assistant threads do not need to check others to print these.

You can write your code and hope it works, however as a computer scientist knowing what your code can and can't reliably do is a commendable trait. As such, we want you to declare what your program does. In the next subsection we will describe this.

Important Note: While this leaving condition ensures an assistant's leaving to not break our original rules, it has a problem. Your program's number of assistants will be exactly two times the number of students, which would make every assistant waiting 3 students to leave impossible as there are not 3 students for each assistant. For example, if you have 2 students and 1 assistant, Even if all the students leave your assistant will be stuck. In this case, you have two options. First is leave it as is. We will ignore deadlocks and missing $leave_a$ lines in your output, but the rest of the conditions will still be checked. Your second option is to change this condition to $\#leave_s \geq 2 * \#leave_a$. If you pick this option your program will not suffer from deadlocks. **Please note that these cases only apply if you decide to implement the leaving condition and not implement the entering condition**.

5.1 Declaring Your Program's Aim

As mentioned earlier, your program needs to declare what conditions it will comply with. The first line your program prints must state the aim of your program. Based on the previous descriptions, you have these options:

- `My program complies with all the conditions.`
- `My program complies with entering and leaving conditions.`
- `My program complies with entering and grouping conditions.`
- `My program complies with leaving and grouping conditions.`
- `My program complies with entering condition.`
- `My program complies with leaving condition.`
- `My program complies with grouping condition.`
- `My program complies with none of the conditions.`

Regardless of what you pick, you need to print all the lines described previously. For example, if you decided to only implement entering and leaving conditions, you still need to print demo lines "I am participating / demo is over" lines.

Important Note: We expect your program to consider any possible scenario that fits the correctness criteria. If you restrict some of the cases, you will lose points. For example if in your code a student can not arrive before an assistant or vice versa you will lose points. The amount of points lost will **depend on how much of the problem you trivialized**. If you knowingly make such a restriction, please explain your restriction inside the **report.pdf**. If you can precisely describe your extra restrictions, you will **lose less points**. If you have extra restrictions, please state this while prompting your programs aim at the first line you print to the console. For instance, if your program satisfies all conditions, but it has some extra restrictions, then your prompt line must be like this:

- My program complies with all the conditions, it has extra restrictions.

6 Useful Information & Tips

- We suggest you start with reading this documentation and planning your solution. There are some subtleties you need to consider, and if you start coding without understanding them, you will most likely try to add them to your already complicated code. This will increase your workload and make a possible debugging harder.

- The solution for this assignment actually is not that complicated. If you take your time to understand how thread synchronization tools like mutexes and barriers work, then plan your solution you can easily prepare your code.
- There are many ways to correctly solve this assignment, but our solution uses
  - one semaphore for entering & leaving conditions.
  - two counters for participants that are waiting for their demo groups and a mutex to protect their atomicity.
  - two semaphores for blocking/waiting & waking up for demo sessions
  - one barrier for making sure 2 students and assistant participated in their demos session

  There are probably more efficient solutions, so we suggest you to find your own solution.
- In order to get familiarity with concurrency problems and how to solve them with semaphores, you can refer to *Little Book of Semaphores*. The book is free and you can find a link on SUCourse.
- We suggest you to test your code regularly. After implementing something new, debugging with breakpoints or using printf statements to be sure its OK up until now is a generally good method when doing homeworks/PAs. If you do so and something goes wrong, you would need to only check the newly added stuff and this will drastically ease your debugging.
- If you get stuck at any point, you can contact your TAs and LAs. They might not answer your emails right away, so we suggest you to ask your questions at the office hours.

Submission

You are expected to submit a zip file named *<YourSUUserName>_PA3.zip* until 18 May 2023, 23:55.

The content of the zip file is as follows:

- **report.pdf**: In your report, you must present the flow of your threads as a pseudo code. You discuss which synchronization mechanisms (semaphores or condition variables or any other logical way) you have chosen, how you implemented, used or adapted them to suit your needs and provide formal arguments on why your code satisfies the correctness criteria described above.

- **demosim.c** or **demosim.cpp**: Your C or C++ implementation file.

If your submission does not fit to the format specified above, your grade may be penalized up to 10 points. So, please pay attention to use zip option when compressing your file (do not use other options like rar or tar), and make sure your report is in pdf, not a txt or word file. Also you need to make sure the name of your submission fits what we asked, if you leave it something like *Cs307_PA3*, your grade will be penalized.

Grading

Some parts of the grading will be automated. If automated tests fail, your code will not be manually inspected for partial points. Some students might be randomly called for an oral exam to explain their implementations and reports.

Submissions will be graded over 100 points:

1. **Compilation (10 pts):** Your program compiles without giving an error. When we run the program, we do not get any runtime errors as well.

2. **Declaring Your Program's Aim (20 pts):** We have defined some ways for you to get partial points in Section 5. You need to declare which of them you aimed for this programming assignment. If your program does what you claimed here you get full points from here. It is really important to know what your code does exactly, and be honest about it. You need to select one of the lines from Section 5.1 to declare correctly. Regardless of the restrictions you choose to implement, your program must create participant threads that agree with the input numbers from the main thread, they must print the lines that we expect from them and the main thread must wait until all the participant threads terminate.

3. **Valid Demo Sessions (20 pts):** Inside the classroom demo prints are as described in the correctness criteria. Threads wait until there is an assistant and two student threads, they all print they are participating, block until other's declaration and the assistant announces that the demo is over.

4. **In Class Condition (30 pts):** The condition where the number of students in the classroom is not more than 3 times the number of assistants inside the classroom holds for any point in execution. This

item is split in two and you can do only one to get 15 points. However, if at any point this inequality does not hold you get 0 points from this part.

- **Entering Condition (15 pts):** Student threads can only enter the classroom if doings so complies with the inequality. You do not need to control participant's leaving.
- **Leaving Condition (15 pts):** Assistants can only leave the classroom if doing so complies with the inequality. You do not need to control participant's entering.

5. **Report (20 pts):** Your report explains your thread methods, how you implemented them, what kind of synchronization mechanisms you used and adapted and why you think that it is correct.

To get points, you need to complete the tasks. Sometimes working and sometimes not is not a valid excuse and will result in getting a 0. For partial points please check section 5.

Important Note: It is possible to solve this problem using only mutexes. In this case, your implementation will suffer from the busy-waiting problem even though you do not use spin-locks. For example if there is only one assistant and three students inside the classroom, when a fourth student enters the classroom it needs to wait in a loop until either an assistant enters or a student leaves. A similar scenario also exists at the demo combinations. Since this is one of the fundamental problems we were trying to avoid in the lectures, you can **lose up to 20 points**, if your implementation suffers from the busy-waiting problem. Note that you do not face such a problem if you use semaphores or combine condition variables with mutexes.

Another Important Note: As explained in Section 5.1, if your program puts extra restrictions that prevents concurrency/possible interleavings we expect you to allow, you will lose some points. However, if you precisely describe these restrictions in your report and declare it in the first line of your program, you will lose less points.

Yet Another Important Note: We will also check deadlocks in your program. If deadlocks are possible, you might lose some points depending on the severity and the complexity of the deadlock.

Sample Output

6.1 Example 1

Input: `demosim 1 2`

```
My program complies with all the conditions.
Thread ID:139650010175040, Role:Assistant, I entered the classroom.
Thread ID:139650018567744, Role:Student, I want to enter the classroom.
Thread ID:139650018567744, Role:Student, I entered the classroom.
Thread ID:139650026960448, Role:Student, I want to enter the classroom.
Thread ID:139650026960448, Role:Student, I entered the classroom.
Thread ID:139650026960448, Role:Student, I am now participating.
Thread ID:139650010175040, Role:Assistant, I am now participating.
Thread ID:139650018567744, Role:Student, I am now participating.
Thread ID:139650010175040, Role:Assistant, demo is over.
Thread ID:139650018567744, Role:Student, I left the classroom.
Thread ID:139650026960448, Role:Student, I left the classroom.
Thread ID:139650010175040, Role:Assistant, I left the classroom.
The Main terminates.
```

Example 2

Input: `demosim 2 4`

```
My program complies with all the conditions.
Thread ID:140737345746496, Role:Student, I want to enter the classroom.
Thread ID:140737337353792, Role:Student, I want to enter the classroom.
Thread ID:140737328961088, Role:Student, I want to enter the classroom.
Thread ID:140737320568384, Role:Student, I want to enter the classroom.
Thread ID:140737312175680, Role:Assistant, I entered the classroom.
Thread ID:140737337353792, Role:Student, I entered the classroom.
Thread ID:140737345746496, Role:Student, I entered the classroom.
Thread ID:140737345746496, Role:Student, I am now participating.
Thread ID:140737312175680, Role:Assistant, I am now participating.
Thread ID:140737303782976, Role:Assistant, I entered the classroom.
Thread ID:140737337353792, Role:Student, I am now participating.
Thread ID:140737337353792, Role:Student, I left the classroom.
Thread ID:140737345746496, Role:Student, I left the classroom.
Thread ID:140737320568384, Role:Student, I entered the classroom.
Thread ID:140737328961088, Role:Student, I entered the classroom.
Thread ID:140737328961088, Role:Student, I am now participating.
Thread ID:140737312175680, Role:Assistant, demo is over.
Thread ID:140737312175680, Role:Assistant, I left the classroom.
Thread ID:140737303782976, Role:Assistant, I am now participating.
Thread ID:140737320568384, Role:Student, I am now participating.
Thread ID:140737328961088, Role:Student, I left the classroom.
Thread ID:140737320568384, Role:Student, I left the classroom.
Thread ID:140737303782976, Role:Assistant, demo is over.
Thread ID:140737303782976, Role:Assistant, I left the classroom.
The main terminates.
```

6.2 Example 3

Input: `demosim 2 4`

```
My program complies with all the conditions.
Thread ID:139745067791936, Role:Student, I want to enter the classroom.
Thread ID:139745051006528, Role:Student, I want to enter the classroom.
Thread ID:139745059399232, Role:Student, I want to enter the classroom.
Thread ID:139745034221120, Role:Assistant, I entered the classroom.
Thread ID:139745067791936, Role:Student, I entered the classroom.
Thread ID:139745025828416, Role:Assistant, I entered the classroom.
Thread ID:139745042613824, Role:Student, I want to enter the classroom.
Thread ID:139745042613824, Role:Student, I entered the classroom.
Thread ID:139745042613824, Role:Student, I am now participating.
Thread ID:139745025828416, Role:Assistant, I am now participating.
Thread ID:139745051006528, Role:Student, I entered the classroom.
Thread ID:139745059399232, Role:Student, I entered the classroom.
Thread ID:139745067791936, Role:Student, I am now participating.
Thread ID:139745067791936, Role:Student, I left the classroom.
Thread ID:139745042613824, Role:Student, I left the classroom.
Thread ID:139745025828416, Role:Assistant, demo is over.
Thread ID:139745025828416, Role:Assistant, I left the classroom.
Thread ID:139745059399232, Role:Student, I am now participating.
Thread ID:139745051006528, Role:Student, I am now participating.
Thread ID:139745034221120, Role:Assistant, I am now participating.
Thread ID:139745034221120, Role:Assistant, demo is over.
Thread ID:139745051006528, Role:Student, I left the classroom.
Thread ID:139745059399232, Role:Student, I left the classroom.
Thread ID:139745034221120, Role:Assistant, I left the classroom.
The Main terminates.
```