CS 3013 - Operating Systems

Project 2 (70 points)
Assigned: Tuesday, January 23, 2024
Checkpoint: Tuesday, January 30, 2024, at 11:59:00pm
Due: Friday, February 2, 2024 at 11:59:00pm

Project 2: Synchronization and Concurrency

C24 Additional Testing: For this assignment, there will be many threads running. If you have been using the Browser VM option, we recommend setting up VirtualBox or UTM and the local Department VM on your machine. This will allow your team to run additional correctness testing. VirtualBox VMs can be "resized" to provide more virtual CPU cores for this assignment, while the Browser VMs currently cannot be resized. This may help with debugging, and surface errors in behavior earlier, but is optional. You will be graded on the correctness of your code.

The following problem will give students a chance to practice synchronization in C programming. The students must create threads for each of the players and debug in a synchronous way. Each thread must use a degree of randomness in delays to test the scheduling. The random numbers should be seeded using `srand()` with the number in a `seed.txt` file (like in Project 1 to set consistent pseudo-randomness). This will allow students to change the seed value to test different scenarios while allowing one to create reproducible scenarios for debugging purposes.

The students must solve this synchronization problem with either (1) mutexes and condition variables (e.g., `pthread_mutex_lock`, `pthread_mutex_unlock`, `pthread_cond_wait`, and `pthread_cond_signal`) OR (2) with semaphores (e.g., `sem_wait`, `sem_post`, and `sem_init`). The students can decide which to use for the problem below. Note that the problem can be solved with either, but one way may be more straightforward than the other. All scheduling must be uncontrolled; students **may not** use a centralized coordinator that chooses which thread runs at any given time.

Problem Description: Worcester Gompei Park (WGP)

What is WGP, one might ask? It is a sports facility that can house multiple types of sports within a single facility. Baseball? Check. Football? Check. Rugby? Them too. With a flip of a few switches (and a bunch of levers), the park can transform into different types of competition fields. This would truly bring Worcester sports efficiency to a new level.

Statement of Work

The WPI CS operating systems professors agreed to assign the problem of distributed sports team coordination to their CS 3013 students. Each project team must construct a simulator for the different sports and players to show that the facility can accomplish the goals. The following constraints were set for the simulator assignment:

1. **Sports Variety and Players:** The WGP needs to support three different types of sports: baseball, football, and rugby. The simulator shall have 36 baseball players (enough for four teams of 9 players), 44 football players (enough for four teams of 11 players), and 60 rugby players (enough for four full teams of 15 players).

2. **Field Maxima and Minima:** To play baseball, the field must have exactly 18 baseball players (9 for each team). If there are not enough baseball players, none should take the field. Any baseball players in excess of 18 must wait until there is another opportunity to play. Likewise, football (the American variety) requires 22 players to be available (11 for each team). While rugby normally requires 30 players (again, 15 for each team), Worcester has agreed to allow team sizes to range from 1 to 15

on the field, so long as teams are evenly matched. Accordingly, rugby can be played with any even number of rugby players up to 30.

3. **Team Safety:** When the field is configured for a particular type of sport, it is considered unsafe for players of a different type of sport to take the field simultaneously. In other words, if the field is set for rugby, no football players or baseball players should be allowed on the field. The same is true for all the sports: only one type of player should use the field at a time.

4. **Maximum Parallelism:** If a pair of rugby players are using the field, any other pairs of rugby players may join the field until the 30 player limit is reached. Teams should try to maximize the number of rugby players unless meeting other constraints (such as fairness, described next). Football and baseball require an exact number of players (i.e., the minimum and maximum are the same), so there is no separate requirement to maximize the player count for those sports.

5. **Fair Performance Opportunities:** Every sports type should have an opportunity to perform without having to wait forever. The rugby players may not deprive the baseball players from playing simply by continuously having a new pair of rugby players join the field every time a pair leaves, thus monopolizing the field. Likewise, baseball players cannot prevent football players from taking the field by colluding to always favor the next batch of baseball players (and the same is true for all other combinations of teams). (In private, the OS programmer types call this the "starvation" constraint, but they were afraid to use that term around the players.)

6. **Game Time Variability and Bounds:** For baseball and football, the players must reach a consensus on the length of the game, but that game length should be selected random. Each pair of rugby players (again, one player on each team) will decide on a time length to play and can leave the field when they are done. When a pair departs, the game proceeds unless the field becomes empty. When a pair of players depart, another pair of players may take their place to maximize parallelism.

7. **Pre-game Napping:** If a player or (or set of players) is ready to go onto the field, they must either immediately enter the field (if permitted under the field limit and constraints #2 (Team Safety) and #3 (Fair Performance Opportunities)) or they must take a nap. Importantly, a player thread may not busy-wait, since that is directly correlated with player fretting and anxiety.

Students must implement code that creates the requisite number of baseball, football, and rugby players thread, with each thread representing a single player. Students must use either semaphores or the combination of mutexes and condition variables to properly synchronize the field. Each baseball, football, or rugby player thread should identify itself (by name or number and player type) and which field slot (e.g., from 1 to the field limit) it is using when it enters or leaves the field. Students should simulate sports activity on the field using the `sleep()` call with a random wait time for the team (or pair, in the case of rugby players). Students should use a random sleep time at the beginning of each thread to emulate players arriving at the WGP at different times. Students may choose to have player compete only twice (a double header) or to return to play continuously until the simulation is manually aborted (e.g., with a Control-C signal). Between each of these competitions, a player should only return to the ready state after waiting a random period of time after their last competition (simulating a rest period).

Students may wonder if they need to keep track of which players are on which teams. To reduce complexity, we do not require students to keep track of teams at all. If they wish to model teams anyway, students can simply assume the first half of the players on the field are associated with Team 1 and the second half are associated with Team 2 and that teams do not persist after the players leave the field.

Students should explain how their solution avoids depriving the different sports player types of the field (avoiding thread starvation) in a text file, `problem_explanation.txt`, that will be submitted along with the source code.

Checkpoint Contributions

Students must submit work that demonstrates substantial progress towards completing the project on the checkpoint date. Substantial progress is judged at the discretion of the grader to allow students flexibility in prioritizing their efforts. However, as an example, any assignment in which three of the seven constraints are implemented will receive full credit towards the checkpoint. **Projects that fail to submit a checkpoint demonstrating significant progress will incur a 10% penalty during final project grading.**

Deliverables and Grading

When submitting the project, please include the following:

- All of the files containing the code.
- The `problem_explanation.txt` explanation for the WGP problem.
- The test files or input (e.g., specific random seed values) that the team used to convince themselves (and others) that the programs actually work.
- Output from the tests showing the system's functionality.
- A document called README.txt explaining the project, any defects, and anything that the team feels the grader should know when grading the project. Only plaintext write-ups are accepted.

Please compress all the files together as a single .zip archive for submission. As with all projects, please only submit standard zip files for compression; **.rar, .7z, and other custom file formats will not be accepted**.

The project programming is only a portion of the project. Students should use the following checklist in turning in their projects to avoid forgetting any deliverables:

1. Sign up for a project partner or have one assigned (URL: `https://ia.wpi.edu/cs3013-taneja/request_teammate.php`). This needs to done for each project separately,
2. Submit the project code and documentation via InstructAssist (URL: `https://ia.wpi.edu/cs3013-taneja/files.php`), and
3. Complete a Partner Evaluation (URL: `https://ia.wpi.edu/cs3013-taneja/evals.php`).

A grading rubric will be provided separately to give a guide for how the project will be graded. No points can be earned for a task that has a prerequisite unless that prerequisite is working well enough to support the dependent task. Students will receive a marked rubric as part of their project grading feedback. Students are expected to contribute equally to their teams; unequal contributions may result in different scores for the students on a team.