## HW 3: Quidditch Synchronization

CS 162

Due: July 13, 2021

**Contents**

1 Introduction 2

2 Setup 2

3 Your Assignment 2

3.1 General Requirements 2

3.2 Functions 3

3.2.1 Initialize 3

3.2.2 Check In 3

3.2.3 Check Out 3

3.2.4 Get Team On Field 4

3.3 Useful Information 4

4 Submission Details 4
1 Introduction

Your task in this assignment is to simulate Quidditch practice at LogPorts Castle. All the Quidditch teams at LogPorts want to use the castle’s Quidditch field for practice, but only one team can use the field at a time.

Your program will need to simulate players arriving at the field and waiting if their team is not already playing on the field.

You will use the synchronization primitives covered in lecture and discussion to implement your program. See the canonical Readers/Writers problem for design and implementation help.

2 Setup

cd ~/code/personal
git pull staff master
cd hw3

You will find a skeleton in checkin.c. checkin.h defines the interface for your check-in system with three functions init_checkin_system, player_checkin, and player_checkout.

You will need to implement these functions. Do not change the names or signatures of these functions, and do not change player.h!

You will also find the file quidditch_test.c, which contains code to load your implementation and run a basic sanity check. Feel free to change this file, as it won’t be graded.

Note: Although this assignment was released on 07/07, please note that the final set of test cases will not be on the autograder until 9:00 AM PST 07/08.

3 Your Assignment

In this section, we’ll describe how the check-in system should work. At a high-level, players must check in to enter the Quidditch field, and check out to exit it.

To implement your check-in system, you will need to modify checkin.c.

3.1 General Requirements

- Each player will always perform four actions in the following order: (1) check in, (2) enter the field, (3) check out, (4) exit the field. See "Functions" for details on how these operations are initiated.

- There can only be one team on the Quidditch field at a time. As a result, players can only enter the field if their team is currently playing on the field.

- There can be any number of teams trying to enter the Quidditch field at any time (many teams from other schools visit LogPorts Castle to practice!). Do NOT assume that there are only 4 Quidditch teams total.

- Each player on the field can only exit after everyone else on the field has also checked out. See "Check Out" for details.
3.2 Functions

This is a summary of the four functions you are responsible for implementing. If you want to see an example of how these functions are used, see `quidditch_test.c`.

3.2.1 Initialize

```c
void init_checkin_system();
```

This function should allocate/initiate any data structures that are necessary for your implementation.

3.2.2 Check In

```c
void player_checkin(pthread_mutex_t *lock, player_t *player);
```

This function simulates a player checking in at the Quidditch field. **Entering this function indicates that the player is checking in. Returning from this function indicates that the player has entered the Quidditch field.**

If there is no team on the field, or the player’s team is already on the field, the function should return without blocking. Otherwise, the function should block until the player’s team is allowed to enter the field (see "Check Out" for details on when a team is allowed to enter the field).

3.2.3 Check Out

```c
void player_checkout(pthread_mutex_t *lock, player_t *player);
```

This function simulates a player checking out from the Quidditch field. **Entering this function indicates that the player is checking out. Returning from this function indicates that the player has exited the Quidditch field.**

The function should block until all other players currently on the field have also called `player_checkout`. If a player is the last player on their team to exit the field, then a new team should be allowed to enter the field (unless there are no waiting players at that time).

Implementation Details:

- You can assume that a thread will call `player_checkout` after `player_checkin`.
- If a new team is allowed to enter the field, the team should be selected in FIFO order. For example, assume that the Gryffindor team is currently playing on the field. Meanwhile, three new players check in to the field in the following order: **Slytherin player, Hufflepuff player, Slytherin player.** After all Gryffindor players exit the field, both Slytherin players should be allowed to enter the field (and the Hufflepuff player should continue waiting).
- Assume there are five Gryffindors playing on the field, and three Slytherin players are waiting to enter the field. **Case I:** If all the five Gryffindor players begin the call to `player_checkout` and subsequently a new Gryffindor player 6 checks in, then Slytherin players should be allowed to take the field before Gryffindor player 6. In other words, Gryffindor player 6 will need to practice separately because they checked in after the other Gryffindor players decided to exit the field. **Case II:** If there exist one or more Gryffindor players who have **not** begun the call to `player_checkout` and subsequently a new Gryffindor player 6 checks in, then Gryffindor player 6 should be allowed to enter the field, and all Gryffindor players must block until all 6 of the Gryffindor players have begun to check out.
3.2.4 Get Team On Field

char *get_team_on_field(pthread_mutex_t *lock);

This function must return the name of the team that is currently playing on the field.

3.3 Useful Information

- You may want to keep track of the following things in your implementation (this is not an exhaustive list):
  - The team that is currently on the field, and the number of players on the field
  - All players that are waiting to enter the field (and their respective teams)
  - The order in which each team checked in to the field
- You are encouraged (but not required) to use PintOS lists. The PintOS lists implementation has been provided as part of the skeleton.
- quidditch_test.c gives you an example of how the functions you implemented could be called to run a simulation. However, the autograder test will likely be different, so you’re encouraged to modify quidditch_test.c and run more complex tests.
- You will notice that a lock is passed in as an argument to the functions player_checkin, player_checkout, and get_team_on_field. You can assume that the calling program will acquire the lock before the functions are called and release the lock after they return. See quidditch_test.c for an example of this.

4 Submission Details

To submit and push to the autograder, first commit your changes, then do:

```
git push personal master
```

Within 30 minutes you should receive an email from the autograder. (If you haven’t received an email within 30 minutes, please notify the instructors via a private post on Piazza.)