**Assignment 2 Documentation**

**Minotaur’s Birthday Party**

**Problem 1**

The Minotaur invited N guests to his birthday party. When the guests arrived, he made

the following announcement.

The guests may enter his labyrinth, one at a time and only when he invites them to do

so. At the end of the labyrinth, the Minotaur placed a birthday cupcake on a plate. When

a guest finds a way out of the labyrinth, he or she may decide to eat the birthday

cupcake or leave it. If the cupcake is eaten by the previous guest, the next guest will find

the cupcake plate empty and may request another cupcake by asking the Minotaur’s

servants. When the servants bring a new cupcake the guest may decide to eat it or leave

it on the plate.

The Minotaur’s only request for each guest is to not talk to the other guests about her or

his visit to the labyrinth after the game has started. The guests are allowed to come up

with a strategy prior to the beginning of the game. There are many birthday cupcakes, so

the Minotaur may pick the same guests multiple times and ask them to enter the

labyrinth. Before the party is over, the Minotaur wants to know if all of his guests have

had the chance to enter his labyrinth. To do so, the guests must announce that they have

all visited the labyrinth at least once.

Now the guests must come up with a strategy to let the Minotaur know that every guest

entered the Minotaur’s labyrinth. It is known that there is already a birthday cupcake left

at the labyrinth’s exit at the start of the game. How would the guests do this and not

disappoint his generous and a bit temperamental host?

**Approach**

* One of the guests will be appointed as the leader, they will eat the cupcake for their first visit and then be in charge of replacing the cupcake in the following visits.
* When a guest enters the labyrinth for the first time, they eat the cupcake.
* For subsequent visits, they leave the cupcake untouched.
* If a guest sees an eaten cupcake, they know that guest has been in the labyrinth before.
* When all guests see an eaten cupcake, they announce that everyone has visited.

**Solution**

The solution for this problem is assigning each guest to a thread and assigning thread #0 be the group leader. The solution for threading was implemented by use of the `mutex` object provided by the c++ standard library. The program will ensure that every guest gets a fair chance to enter the labyrinth and potentially eat the cupcake. It does this by using a random selection process as the Minotaur would have called them out in order to determine which guest gets to move next. This randomness ensures that no guest is favored over others, making the game fair for everyone involved.

To prevent any unfair advantage or confusion, the program uses locks (mutexes) to make sure that only one guest can access critical information at a time. For example, when a guest is deciding whether to eat the cupcake, no other guest can interfere or change the state of the cupcake. This helps maintain order and prevents any cheating or unfair actions.The group leader keeps track of how many guests have entered the labyrinth. The game will continue running until all guests have had their turn. Once every guest has entered the program exits. This program outputs which guest has eaten the cupcake and announces when all guests have eaten the cupcake.

**Runtime**

Below are a few runtime results for the program under different number of threads:

```

30 threads: 1.118 seconds

50 threads: 4.034 seconds

100 threads: 17.802 seconds

200 threads: 90.726 seconds

```

**Output**

**```**

**Guest #0 ate the cupcake!**

**Guest #1 ate the cupcake!**

**Guest #4 ate the cupcake!**

**Guest #2 ate the cupcake!**

**Guest #3 ate the cupcake!**

**All 5 guests entered the labyrinth.**

**Game finished in 3 milliseconds**

**```**

**Correctness**

The algorithm appoints a leader and assigns each guest to a thread. To ensure fairness, it uses random selection for guests to take turns entering the labyrinth and potentially eating the cupcake. Critical information like the cupcake's state is protected using locks, ensuring only one guest can access it at a time. The leader keeps track of guests entering, and the game continues until everyone has had their turn. The leader manages the game's progress, and the algorithm provides clear feedback to indicate when the game is finished. By combining these elements, the algorithm effectively solves the problem while maintaining fairness and integrity throughout the process.

**Problem 2**

The Minotaur decided to show his favorite crystal vase to his guests in a dedicated

showroom with a single door. He did not want many guests to gather around the vase

and accidentally break it. For this reason, he would allow only one guest at a time into

the showroom. He asked his guests to choose from one of three possible strategies for

viewing the Minotaur’s favorite crystal vase:

1) Any guest could stop by and check whether the showroom’s door is open at any time and try to enter the room. While this would allow the guests to roam around the castle

and enjoy the party, this strategy may also cause large crowds of eager guests to gather

around the door. A particular guest wanting to see the vase would also have no

guarantee that she or he will be able to do so and when.

2) The Minotaur’s second strategy allowed the guests to place a sign on the door

indicating when the showroom is available. The sign would read “AVAILABLE” or

“BUSY.” Every guest is responsible to set the sign to “BUSY” when entering the

showroom and back to “AVAILABLE” upon exit. That way guests would not bother trying

to go to the showroom if it is not available.

3) The third strategy would allow the quests to line in a queue. Every guest exiting the

room was responsible to notify the guest standing in front of the queue that the

showroom is available. Guests were allowed to queue multiple times.

Which of these three strategies should the guests choose? Please discuss the advantages

and disadvantages.

**Approach**

I chose **Strategy 2: Sign-Based Availability** for this solution is similar to that of the birthday party problem except that there will be no leader to count or replace the cupcakes.

**Advantages**

* Guests can easily see whether the showroom is \*\*available\*\* or \*\*busy\*\*.
* Reduces unnecessary attempts to enter when the room is occupied.
* It strikes a balance between simplicity and efficiency.
* Guests can focus on enjoying the party without worrying about missing their turn.

**Disadvantages**:

* In a real life situation the guests might forget to change the sign, leading to confusion.
* If many guests exit simultaneously, there could be a queue forming outside.

**Solution**

Create N threads and a set that keeps track of which guests have already seen the vase. When a guest "enters" the room, status is set to busy, admires the vase for a random amount of time simulated by a random generator function then sets the status back to available. After a guest is done admired the vase, theirID is added to the set. The output of the program is status of who is currently viewing the vase.

An example run of the algorithm with 3 guests:

1. \*\*Guest 1\*\*:

- Enters the showroom.

- Sets the sign to \*\*"BUSY"\*\*.

- Views the vase.

- Exits and changes the sign to \*\*"AVAILABLE"\*\*.

2. \*\*Guest 2\*\*:

- Checks the sign: \*\*"AVAILABLE"\*\*.

- Enters the showroom.

- Sets the sign to \*\*"BUSY"\*\*.

- Views the vase.

- Exits and changes the sign to \*\*"AVAILABLE"\*\*.

3. \*\*Guest 3\*\*:

- Checks the sign: \*\*"AVAILABLE"\*\*.

- Enters the showroom.

- Sets the sign to \*\*"BUSY"\*\*.

- Views the vase.

- Exits and changes the sign to \*\*"AVAILABLE"\*\*.

**Runtime**

I tested the runtime of the program taking the average runtime over `10` trials. Below are the runtime results of the program for different number of threads:

```

30 threads: 2.884 seconds

50 threads: 4.718 seconds

100 threads: 9.537 seconds

200 threads: 18.955 seconds

```

**Output**

```

Guest #0 is viewing the vase

Guest #2 is viewing the vase

Guest #3 is viewing the vase

Guest #4 is viewing the vase

Guest #1 is viewing the vase

All 5 guests have viewed the vase.

Finished in 483 milliseconds

```

**Correctness**

The algorithm maintains correctness by enforcing exclusive access to the vase through the "BUSY" status mechanism, guaranteeing that guests don't interrupt each other's viewing sessions. Additionally, the output of the program accurately reflects the current status of who is viewing the vase, providing clear visibility into the process. Therefore, the algorithm ensures the correct and orderly viewing of the vase by each guest, preventing any conflicts and providing status updates on the current viewer of the vase throughout the process.