

CS361 - Thread Race Answer Key

Generated by Doxygen 1.9.4

1 CS361 - Thread Race 2000	1
1.1 Description	1
2 Class Index	3
2.1 Class List	3
3 File Index	5
3.1 File List	5
4 Class Documentation	7
4.1 DiceQueue Class Reference	7
4.1.1 Detailed Description	7
4.1.2 Constructor & Destructor Documentation	7
4.1.2.1 DiceQueue()	7
4.1.2.2 ~DiceQueue()	8
4.1.3 Member Function Documentation	8
4.1.3.1 addDice()	8
4.1.3.2 getDice()	8
4.1.4 Member Data Documentation	8
4.1.4.1 dice	8
4.1.4.2 items	8
4.1.4.3 lock	9
4.2 LeaderBoard Class Reference	9
4.2.1 Detailed Description	9
4.2.2 Constructor & Destructor Documentation	9
4.2.2.1 LeaderBoard()	9
4.2.2.2 ~LeaderBoard()	10
4.2.3 Member Function Documentation	10
4.2.3.1 addRacer()	10
4.2.3.2 getPlace()	10
4.2.3.3 raceFinished()	11
4.2.4 Member Data Documentation	11
4.2.4.1 current	11
4.2.4.2 racers	11
4.2.4.3 results	11
4.2.4.4 s	11
4.3 semaphore Class Reference	12
5 File Documentation	13
5.1 src/dicequeue.cpp File Reference	13
5.1.1 Detailed Description	13
5.1.2 Description	13
5.2 src/dicequeue.h File Reference	13
5.2.1 Detailed Description	14

5.2.2 Description	14
5.3 dicequeue.h	14
5.4 src/leaderboard.cpp File Reference	14
5.4.1 Detailed Description	14
5.4.2 Description	15
5.5 src/leaderboard.h File Reference	15
5.5.1 Detailed Description	15
5.5.2 Description	15
5.6 leaderboard.h	15
5.7 src/main.cpp File Reference	16
5.7.1 Detailed Description	16
5.7.2 Description	16
5.7.3 Function Documentation	16
5.7.3.1 checkPosInt()	16
5.7.3.2 gameMaster()	17
5.7.3.3 main()	17
5.7.3.4 moveMessage()	17
5.7.3.5 racer()	18
5.7.3.6 randomSleep()	18
5.7.3.7 startMessage()	18
5.7.3.8 stopMessage()	18
5.7.4 Variable Documentation	18
5.7.4.1 ioLock	19
5.8 semaphore.h	19
Index	21

Chapter 1

CS361 - Thread Race 2000

1.1 Description

This program simulates threads racing. It requires the programmer to make safe data structures or protect existing data structures.

Make Commands:

`make`

will build the executable.

`make run`

will run the experiments.

`make clean`

will clear out the compiled code.

`make doc`

will build the doxygen files.

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

DiceQueue	7
LeaderBoard	9
semaphore	12

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

src/dicequeue.cpp	13
src/dicequeue.h	13
src/leaderboard.cpp	14
src/leaderboard.h	15
src/main.cpp	16
src/semaphore.h	19

Chapter 4

Class Documentation

4.1 DiceQueue Class Reference

```
#include <dicequeue.h>
```

Public Member Functions

- [DiceQueue](#) ()
- [~DiceQueue](#) ()
- void [addDice](#) (int d)
- int [getDice](#) ()

Private Attributes

- std::queue< int > * [dice](#)
- semaphore * [items](#)
- semaphore * [lock](#)

4.1.1 Detailed Description

Implement a Queue using the Producer-Consumer Problem.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 DiceQueue()

```
DiceQueue::DiceQueue ( )
```

Create a new empty Queue. Set up semaphores correctly.

4.1.2.2 ~DiceQueue()

```
DiceQueue::~~DiceQueue ( )
```

Delete all allocated objects.

4.1.3 Member Function Documentation

4.1.3.1 addDice()

```
void DiceQueue::addDice (
    int d )
```

Add a dice roll to the queue when it is safe to do so.

Parameters

<i>d</i>	is the dice roll to add.
----------	--------------------------

4.1.3.2 getDice()

```
int DiceQueue::getDice ( )
```

Get a dice roll or wait until one is available.

Returns

A roll of the dice

4.1.4 Member Data Documentation

4.1.4.1 dice

```
std::queue<int>* DiceQueue::dice [private]
```

The actual queue of dice

4.1.4.2 items

```
semaphore* DiceQueue::items [private]
```

Semaphore to signal consumers that the producer has produced.

4.1.4.3 lock

```
semaphore* DiceQueue::lock [private]
```

Semaphore to protect the queue.

The documentation for this class was generated from the following files:

- [src/dicequeue.h](#)
- [src/dicequeue.cpp](#)

4.2 LeaderBoard Class Reference

```
#include <leaderboard.h>
```

Public Member Functions

- [LeaderBoard](#) (int numRacers)
- [~LeaderBoard](#) ()
- void [addRacer](#) (std::thread::id me)
- bool [raceFinished](#) ()
- std::thread::id [getPlace](#) (int i)

Private Attributes

- int [racers](#)
- std::thread::id * [results](#)
- int [current](#)
- [semaphore](#) * [s](#)

4.2.1 Detailed Description

The [LeaderBoard](#) records which order the threads finish in.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 LeaderBoard()

```
LeaderBoard::LeaderBoard (  
    int numRacers )
```

Create a new Leader Board with given number of spaces

Parameters

<i>numRacers</i>	is the total number of racers playing
------------------	---------------------------------------

4.2.2.2 ~LeaderBoard()

```
LeaderBoard::~~LeaderBoard ( )
```

Free the class from memory

4.2.3 Member Function Documentation**4.2.3.1 addRacer()**

```
void LeaderBoard::addRacer (
    std::thread::id me )
```

Add a racer to the leaderboard. Position is determined automatically.

Parameters

<i>me</i>	is the ID of the thread that just finished
-----------	--

4.2.3.2 getPlace()

```
std::thread::id LeaderBoard::getPlace (
    int i )
```

Find out which thread crossed the finish line in position i.

Parameters

<i>i</i>	is the position to check
----------	--------------------------

Returns

The thread ID of the racer in that position

4.2.3.3 raceFinished()

```
bool LeaderBoard::raceFinished ( )
```

Determine if the race is finished

Returns

True if all racers have crossed the finish line

4.2.4 Member Data Documentation

4.2.4.1 current

```
int LeaderBoard::current [private]
```

position of next racer to finish

4.2.4.2 racers

```
int LeaderBoard::racers [private]
```

number of racers playing

4.2.4.3 results

```
std::thread::id* LeaderBoard::results [private]
```

Array to store results of race

4.2.4.4 s

```
semaphore* LeaderBoard::s [private]
```

Semaphore to protect the attributes

The documentation for this class was generated from the following files:

- [src/leaderboard.h](#)
- [src/leaderboard.cpp](#)

4.3 semaphore Class Reference

Public Member Functions

- **semaphore** (unsigned int size)
- void **signal** ()
- void **wait** ()

Private Attributes

- unsigned int **counter**
- std::mutex **myLock**
- std::condition_variable **myCond**

The documentation for this class was generated from the following files:

- src/semaphore.h
- src/semaphore.cpp

Chapter 5

File Documentation

5.1 src/dicequeue.cpp File Reference

```
#include "dicequeue.h"
```

5.1.1 Detailed Description

Author

Mark Boady

Date

2021-2022

5.1.2 Description

Implementation of the Dice Queue Class

5.2 src/dicequeue.h File Reference

```
#include "semaphore.h"  
#include <queue>
```

Classes

- class [DiceQueue](#)

5.2.1 Detailed Description

Author

Mark Boady

Date

2021-2022

5.2.2 Description

The dice queue is used to send dice from the game master to the players. It uses a solution to the producer-consumer problem.

5.3 dicequeue.h

[Go to the documentation of this file.](#)

```
1
10 #ifndef _DICE_QUEUE_H_
11 #define _DICE_QUEUE_H_
12
13 #include "semaphore.h"
14 #include <queue>
15
19 class DiceQueue{
20 private:
21     std::queue<int>* dice;
22     semaphore* items;
23     semaphore* lock;
24 public:
28     DiceQueue();
32     ~DiceQueue();
37     void addDice(int d);
42     int getDice();
43 };
44
45 #endif
```

5.4 src/leaderboard.cpp File Reference

```
#include "leaderboard.h"
```

5.4.1 Detailed Description

Author

Mark Boady

Date

2021-2022

5.4.2 Description

Implementation of the [LeaderBoard](#) Class.

5.5 src/leaderboard.h File Reference

```
#include <thread>
#include "semaphore.h"
```

Classes

- class [LeaderBoard](#)

5.5.1 Detailed Description

Author

Mark Boady

Date

2021-2022

5.5.2 Description

A leaderboard is used to track when each racer finishes the race.

5.6 leaderboard.h

[Go to the documentation of this file.](#)

```
1
9 #ifndef _LEADER_BOARD_H_
10 #define _LEADER_BOARD_H_
11
12 #include <thread>
13 #include "semaphore.h"
14
18 class LeaderBoard{
19 private:
20     int racers;
21     std::thread::id* results;
22     int current;
23     semaphore* s;
24 public:
29     LeaderBoard(int numRacers);
33     ~LeaderBoard();
38     void addRacer(std::thread::id me);
43     bool raceFinished();
49     std::thread::id getPlace(int i);
50 };
51
52 #endif
```

5.7 src/main.cpp File Reference

```
#include <iostream>
#include <string>
#include <thread>
#include <mutex>
#include "semaphore.h"
#include "dicequeue.h"
#include "leaderboard.h"
```

Functions

- bool [checkPosInt](#) (std::string text)
- void [randomSleep](#) ()
- void [startMessage](#) ()
- void [moveMessage](#) (int step)
- void [stopMessage](#) ()
- void [gameMaster](#) (DiceQueue *DQ, LeaderBoard *LB)
- void [racer](#) (DiceQueue *DQ, LeaderBoard *LB)
- int [main](#) (int argc, char **argv)

Variables

- std::mutex [ioLock](#)

5.7.1 Detailed Description

Author

Mark Boady

Date

2021-2022

5.7.2 Description

Answer key example for Thread Race 2000 Homework

5.7.3 Function Documentation

5.7.3.1 checkPosInt()

```
bool checkPosInt (
    std::string text )
```

Check that a given string contains a positive integer

Parameters

<i>text</i>	is the string to examine
-------------	--------------------------

Returns

true if the string is a positive integer

5.7.3.2 gameMaster()

```
void gameMaster (
    DiceQueue * DQ,
    LeaderBoard * LB )
```

The Game Master thread rolls dice and adds them to a queue. It runs until the leader board is full.

Parameters

<i>DQ</i>	is a safe queue to put the dice into
<i>LB</i>	is a safe leaderboard to record results

5.7.3.3 main()

```
int main (
    int argc,
    char ** argv )
```

Create racers and game master. Simulate a game being played.

Parameters

<i>argc</i>	must be 2
<i>argv</i>	contains the number of racers in argv[1]

Returns

0 on success and 1 on error

5.7.3.4 moveMessage()

```
void moveMessage (
    int step )
```

Print that a thread has moved forward

Parameters

<i>step</i>	is how far forward the thread moved
-------------	-------------------------------------

5.7.3.5 racer()

```
void racer (
    DiceQueue * DQ,
    LeaderBoard * LB )
```

The racer takes dice rolls until it has moved 20 spaces

Parameters

<i>DQ</i>	is a safe queue to read dice from
<i>LB</i>	is a safe leaderboard to record when I cross the finish line

5.7.3.6 randomSleep()

```
void randomSleep ( )
```

Sleep thread for a random length of time between 0 and 2 seconds

5.7.3.7 startMessage()

```
void startMessage ( )
```

Print that a thread has started

5.7.3.8 stopMessage()

```
void stopMessage ( )
```

Print that a thread has crossed the finish line

5.7.4 Variable Documentation

5.7.4.1 ioLock

```
std::mutex ioLock
```

Global Lock to Protect I/O

5.8 semaphore.h

```
1 //Mark Boady Drexel 2021
2 //A semaphore class using locks and condition variables
3 //Used to make an example of the Rendezvous
4 //problem from the Little Book of Semaphores
5
6 #ifndef _SEMAPHORE_H_
7 #define _SEMAPHORE_H_
8
9 #include <mutex>
10 #include <condition_variable>
11
12
13 class semaphore{
14     private:
15         //The counter for increment and decrement
16         unsigned int counter;
17         //Mutex to lock access to counter
18         //this should be mutable you will read why
19         mutable std::mutex myLock;
20         //Condition Variable for waits
21         std::condition_variable myCond;
22     public:
23         //Default to 1 if no value given
24         semaphore();
25         //User gives the count
26         semaphore(unsigned int size);
27         //signal other threads
28         void signal();
29         //Wait for signal
30         void wait();
31 };
32
33 #endif
```


Index

- ~DiceQueue
 - DiceQueue, [7](#)
- ~LeaderBoard
 - LeaderBoard, [10](#)
- addDice
 - DiceQueue, [8](#)
- addRacer
 - LeaderBoard, [10](#)
- checkPosInt
 - main.cpp, [16](#)
- current
 - LeaderBoard, [11](#)
- dice
 - DiceQueue, [8](#)
- DiceQueue, [7](#)
 - ~DiceQueue, [7](#)
 - addDice, [8](#)
 - dice, [8](#)
 - DiceQueue, [7](#)
 - getDice, [8](#)
 - items, [8](#)
 - lock, [8](#)
- gameMaster
 - main.cpp, [17](#)
- getDice
 - DiceQueue, [8](#)
- getPlace
 - LeaderBoard, [10](#)
- ioLock
 - main.cpp, [18](#)
- items
 - DiceQueue, [8](#)
- LeaderBoard, [9](#)
 - ~LeaderBoard, [10](#)
 - addRacer, [10](#)
 - current, [11](#)
 - getPlace, [10](#)
 - LeaderBoard, [9](#)
 - raceFinished, [10](#)
 - racers, [11](#)
 - results, [11](#)
 - s, [11](#)
- lock
 - DiceQueue, [8](#)
- main
 - main.cpp, [17](#)
- main.cpp
 - checkPosInt, [16](#)
 - gameMaster, [17](#)
 - ioLock, [18](#)
 - main, [17](#)
 - moveMessage, [17](#)
 - racer, [18](#)
 - randomSleep, [18](#)
 - startMessage, [18](#)
 - stopMessage, [18](#)
- moveMessage
 - main.cpp, [17](#)
- raceFinished
 - LeaderBoard, [10](#)
- racer
 - main.cpp, [18](#)
- racers
 - LeaderBoard, [11](#)
- randomSleep
 - main.cpp, [18](#)
- results
 - LeaderBoard, [11](#)
- s
 - LeaderBoard, [11](#)
- semaphore, [12](#)
- src/dicequeue.cpp, [13](#)
- src/dicequeue.h, [13](#), [14](#)
- src/leaderboard.cpp, [14](#)
- src/leaderboard.h, [15](#)
- src/main.cpp, [16](#)
- src/semaphore.h, [19](#)
- startMessage
 - main.cpp, [18](#)
- stopMessage
 - main.cpp, [18](#)