

# COMP 2710 – Spring 2022

## Final Exam Project

Points Possible: 50

**There should be no collaboration among students.** A student should not share any code with other students. Collaborations among students in any form will be treated as a serious violation of the University's academic integrity code.

### Objectives:

1. Complete the source code to simulate the dining-philosopher problem.
2. Understand the basic concepts of the deadlock principal.
3. Build a binary program on a Linux machine.
4. Run the program and record the results of the simulation.

### Requirements:

- Each student should **independently** accomplish the project.
- **Important!** Read and follow the I/O format specification carefully. It is very important to your final grade of this project!
- You are highly recommended to use a Linux operating system.

### 1. Introduction to dining philosopher model

Dining philosopher problem is a model to schedule concurrent processes and threads as much as possible while avoid the deadlock. It contains:

1. 5 Philosophers: Represent processes or threads that consume the resources.
2. 5 Chopsticks: Represent 5 units of the resources.

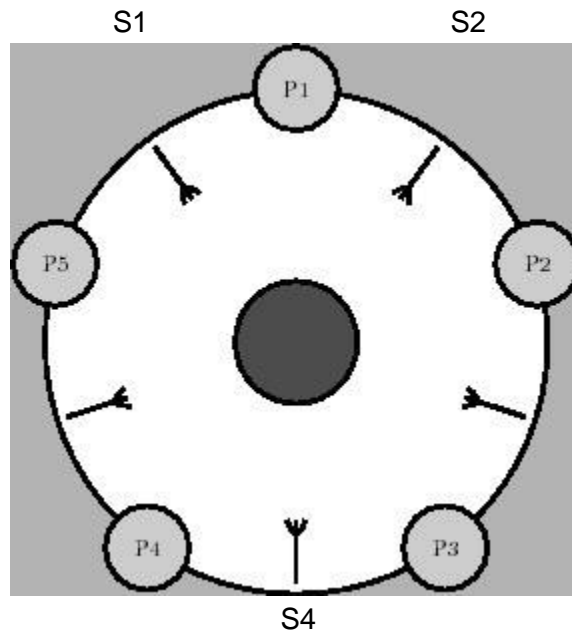
### Other concepts involved in our project:

3. Mutex: a "lock" guarantees that only one person has access to the resource.
4. Semaphore: represents the resource "capacity".

The mutex only has 1 capacity, 0 or 1, but Semaphore can have more than one capacity. In this project, we use POSIX threads. The "pthread" is a POSIX thread library to provide the basic functions.

### Description:

There are 5 philosophers sitting around a table and try to eat from the center of the table. There are also 5 chopsticks on the table. Let us call the 5 philosophers in clockwise P1, P2, ..., P5. There are also 5 chopsticks clockwise S1, S2, ..., S5 on the table. There is only one chopstick between every two philosophers. Every right-hand chopstick has the same index as the index of the philosopher. For example, on the right-hand side of P1, the chopstick is called S1. And every left-hand side chopstick has the index (1 + number) of the philosopher. For example, the left-hand side of P1 is the chopstick S2. See the picture below:



1. A philosopher spends a random time to think, then (s)he feels hungry and try to eat.
2. The middle dish can provide enough food for everyone at the same time.
3. A philosopher only can start to eat when he picks up two chopsticks from left hand side and right-hand side to form a pair of chopsticks.
4. If a philosopher takes one chopstick, (s)he will try to fight with neighbors to get another one, and never back off to put down the one in their hands.
5. You cannot interrupt a philosopher when they are eating.

Here comes the deadlock problem: when all the 5 philosophers start to pick up the right-hand side chopsticks at the same time, they get stuck.

Here we can use a semaphore to avoid this. Whenever one philosopher starts to pick chopsticks, the semaphore is set to 1, and he is the only one that can pick chopsticks.

See the example I/O format:

```
[tmp0038@tux250:~/FinalExam$ ./FinalExam_Pharm_tmp0038.out
Banner id: 903900281
Philosopher 0 is thinking.
Philosopher 1 is thinking.
Philosopher 3 is thinking.
Philosopher 4 is thinking.
Philosopher 2 is thinking.
Philosopher 1 is hungry.
Philosopher 1 picked up Chopstick 1, can NOT eat with one chopstick.
Philosopher 1 picked up Chopstick 2, now is eating with two chopsticks.
Philosopher 2 is hungry.
Philosopher 0 is hungry.          P0 needs to wait for chopstick 1 since P1 is using it
Philosopher 0 picked up Chopstick 0, can NOT eat with one chopstick.
Philosopher 3 is hungry.
Philosopher 3 picked up Chopstick 3, can NOT eat with one chopstick.
Philosopher 3 picked up Chopstick 4, now is eating with two chopsticks.
Philosopher 3 put down Chopstick 3
Philosopher 3 put down Chopstick 4
Philosopher 3 is thinking.
Philosopher 4 is hungry.
Philosopher 4 picked up Chopstick 4, can NOT eat with one chopstick.
Philosopher 1 put down Chopstick 1  P1 finished eating → release chopstick 1
Philosopher 1 put down Chopstick 2
Philosopher 1 is thinking.          P0 has picked up 2 chopsticks and started eating
Philosopher 0 picked up Chopstick 1, now is eating with two chopsticks.
Philosopher 2 picked up Chopstick 2, can NOT eat with one chopstick.
Philosopher 2 picked up Chopstick 3, now is eating with two chopsticks.
Philosopher 2 put down Chopstick 2
Philosopher 2 put down Chopstick 3
```

Your results may be different from above due to randomness.

As you learned from slides, this is not good enough, because the food is enough for everyone. We try to make as many as possible philosophers to eat at the same time!

Currently, the five philosophers form a cycle. To break it, we can allow up to 4 philosophers to pick chopsticks at the same time. Four philosophers can never form a cycle so no deadlock will happen.

## 2. Follow the Format Specification

In the source file " FinalExam\_Coding.cpp", you will find the first four comment lines:

```
// #0#BEGIN# DO NOT MODIFY THIS COMMENT LINE!
// Firstname
// Lastname
// #0#END# DO NOT MODIFY THIS COMMENT LINE!
```

Your first task is modifying the two lines **between** the beginning line and end line. Change them into your first name and last name. Remember the strings are case-sensitive, so capitalize the first letter of the word and follow exactly the syntax of the example.

You can see lots of similar code blocks in the file. You are free and supposed to fill your answers between those special beginning and ending comment lines. You can insert and edit multiple lines between special comment lines in anyways, however (**Important!**), as the comment indicated, do not modify the special begin and comment lines **themselves!**

Let's do the second task. Scroll down to the bottom of the file and find those lines (press "shift + g" if you are using vi/vim):

```
// #6#BEGIN# DO NOT MODIFY THIS COMMENT LINE!
int banner_id = 123456789;
// #6#END# DO NOT MODIFY THIS COMMENT LINE!
```

Change the number to **your own ID**. Your unique student ID will be compiled into the program, and the input of the experiment also uniquely depends on your ID to prevent identical names (It happened last semester).

**Warning:** Since every student has a unique id number, the later compiled binary file is also unique. Copy binary file from other students will be easily detected!

### 3. Complete Source Code (50 Points. 5 points each. Not include #0#)

Read the source code "Question\_FinalExam\_Official.cpp", try to understand the function of each line of code, the basic usage of ``pthread`` library function and semaphore.

Follow the instructions in the comments Question\_FinalExam\_Official.cpp and insert proper code into the rest **10** blocks to implement the producer/consumer model.

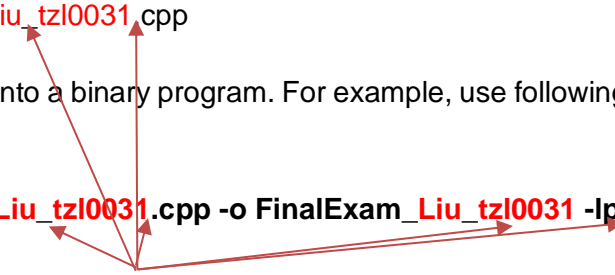
### 4. Run and Record Simulation Result

Your correct file name should be: **FinalExam\_LastName\_UserID.cpp**

For example, FinalExam\_**Liu\_tzl0031**.cpp

Compile your source code into a binary program. For example, use following command to include the ``pthread`` library:

```
$ g++ FinalExam_Liu_tzl0031.cpp -o FinalExam_Liu_tzl0031 -lpthread
```



Change it because it  
is TA's last name  
and ID

After you compile the C++ source code successfully, please use the script command to record the running result of your program:

```
$ script FinalExam_Liu_tzl0031.script
Script started, file is FinalExam_Liu_tzl0031.script
```

```
./FinalExam_Liu_tzl0031
```

After you run the program, you will have the following results:

```
Banner id: 123456789
Philosopher 0 is thinking
...
```

You should have similar result and difference in the banner ID. Then exit recording and save it into typescript file "FinalExam\_Pham\_tmp0038.script" by the following command:

```
$ exit
exit Script done, file is FinalExam_Liu_tzl0031.script
```

**Warning:** Since every student has a unique id number, the result of the simulation is also unique. Copy simulation results from other students will be easily detected!

## 5. Deliverables

Until now, you have generated multiple files. Create a folder and make sure all three files are in the folder. You should have those following files inside the folder:

1. Commands recording file: FinalExam\_LastName\_UserID.script
2. Executable binary file: FinalExam\_LastName\_UserID
3. Source code file: FinalExam\_LastName\_UserID.cpp

Change the folder name to "FinalExam\_LastName\_UserID". You can use the command **mv** to rename the folder:

```
$ mv your-current-folder-name FinalExam_LastName_UserID
```

Achieve all folders into a single tarred and compressed file with a tar command.

```
tar -zcvf FinalExam_LastName_UserID.tar.gz FinalExam_LastName_UserID
```

You need to submit one tarred file with a format: **FinalExam\_LastName\_UserID.tar.gz**