Lecturers:	Prof. Dr. Florina M. Ciorba	florina.ciorba@unibas.ch
	Dr. Ahmed Eleliemy	ahmed.eleliemy@unibas.ch
Assistants:	Thomas Jakobsche	thomas.jakobsche@unibas.ch
Tutors:	Selaudin Agolli	s.agolli@unibas.ch
	Olivier Heinz Mattman	olivier.mattmann@unibas.ch
	Agni Ramadani	agni.ramadani@unibas.ch

Exercise 3: Synchronization (10 points)

Given: April 08, 2022
Deadline: May 03, 2022

Objectives

- Understand the producer-consumer problem (semaphores and mutex locks).
- Investigate the dining-philosopher problem and report on the output.
- Correct the dining-philosopher problem and report the correct output.
- Investigate and correct example code that contains problems similar to deadlocks.

Instructions

- You can solve this exercises in teams of two.
- Submit the solution of each task with detailed comments that clarify your solution.
- Show your solution and upload it to https://adam.unibas.ch.
- Provide all deliverables as an archive file.

Task 1: Bounded-Buffer and Producer-Consumer **(3 points)**

In this task you will work on the bounded-buffer problem using the producer-consumer model. Producers and consumers (running as separate threads) move items to and from a buffer with a fixed size. T1.c contains the code without the necessary synchronization.

Hint: In this bounded-buffer example producers should stop producing when the buffer is full, and consumers should only consume items that are actually in the buffer.

To compile the code: **gcc -o T1 T1.c -lpthread**
To execute the code: **./T1 <duration> <producer threads> <consumer threads>**

i) Execute T1.c with the parameters below, report the output and explain the problems.

- ./T1 10 5 0
- ./T1 10 0 5

ii) Correct the code by inserting the necessary synchronization, execute your corrected code with the parameters below, report the output and explain the correct process of the producer-consumer model. **Hint:** You can use counting semaphores and mutex locks.

- ./T1 10 5 0
- ./T1 10 0 5
- ./T1 10 2 2

You must use the given source file T1.c as your starting point. All you need is to implement the open TODOs in the code.

Task 2: Dining-Philosopher **(3 points)**

In this task you will work on the dining-philosophers problem using condition variables. Philosophers spend their lives alternating between thinking and eating, thinking and eating, etc. They occasionally try to pick up forks to eat from a bowl at the center of the table. They can only eat when their neighbors are not eating.

Hint: If you do not see the "DINNER IS OVER" message at the end of the program, then something is wrong and your code might encounter a deadlock. Deadlocks might not always occur, so try to run your code multiple times to be sure.

To compile the code: **make all**
To execute the code: **./diningphilosophers**

There are multiple files in this task. All you need is to implement the open TODOs in the code (main.c and dining.c).

Task 3: Problem Investigation (2 points)

Investigate the code example given below, in Listing 1. What is the name of the problem and how can you solve it?

Listing 1: problem example

```
1  // thread one runs in this function
2  void *do_work_one(void *param)
3  {
4      int done = 0;
5      while (!done)
6      {
7          pthread_mutex_lock(&first_mutex);
8          if (pthread_mutex_trylock(&second_mutex))
9          {
10             // do some work
11             pthread_mutex_unlock(&second_mutex);
12             pthread_mutex_unlock(&first_mutex);
13             done = 1;
14         }
15         else
16             pthread_mutex_unlock(&first_mutex);
17     }
18     pthread_exit(0);
19 }
20
21 // thread two runs in this function
22 void *do_work_two(void *param)
23 {
24     int done = 0;
25     while (!done)
26     {
27         pthread_mutex_lock(&second_mutex);
28         if (pthread_mutex_trylock(&first_mutex))
29         {
30             // do some work
31             pthread_mutex_unlock(&first_mutex);
32             pthread_mutex_unlock(&second_mutex);
33             done = 1;
34         }
35         else
36             pthread_mutex_unlock(&second_mutex);
37     }
38     pthread_exit(0);
39 }
```

Task 4: Synchronization Problems (1 point)

Describe the classical synchronization problems and tools to solve them.

Task 5: Deadlock vs. Starvation (1 point)

Describe the difference between deadlocks and starvation.